

Introduction to Microservice using Moleculer Framework

Go Frendi Gunawan,

Lecturer at STIKI Malang,
Backend Engineer at Kata.ai

Before We Start

- Goal
 - You will understand what microservice is/is not
 - You know how microservice works
 - You can implement minimal microservice using `moleculer.js`
- Non Goal
 - Building enterprise application

Architecture

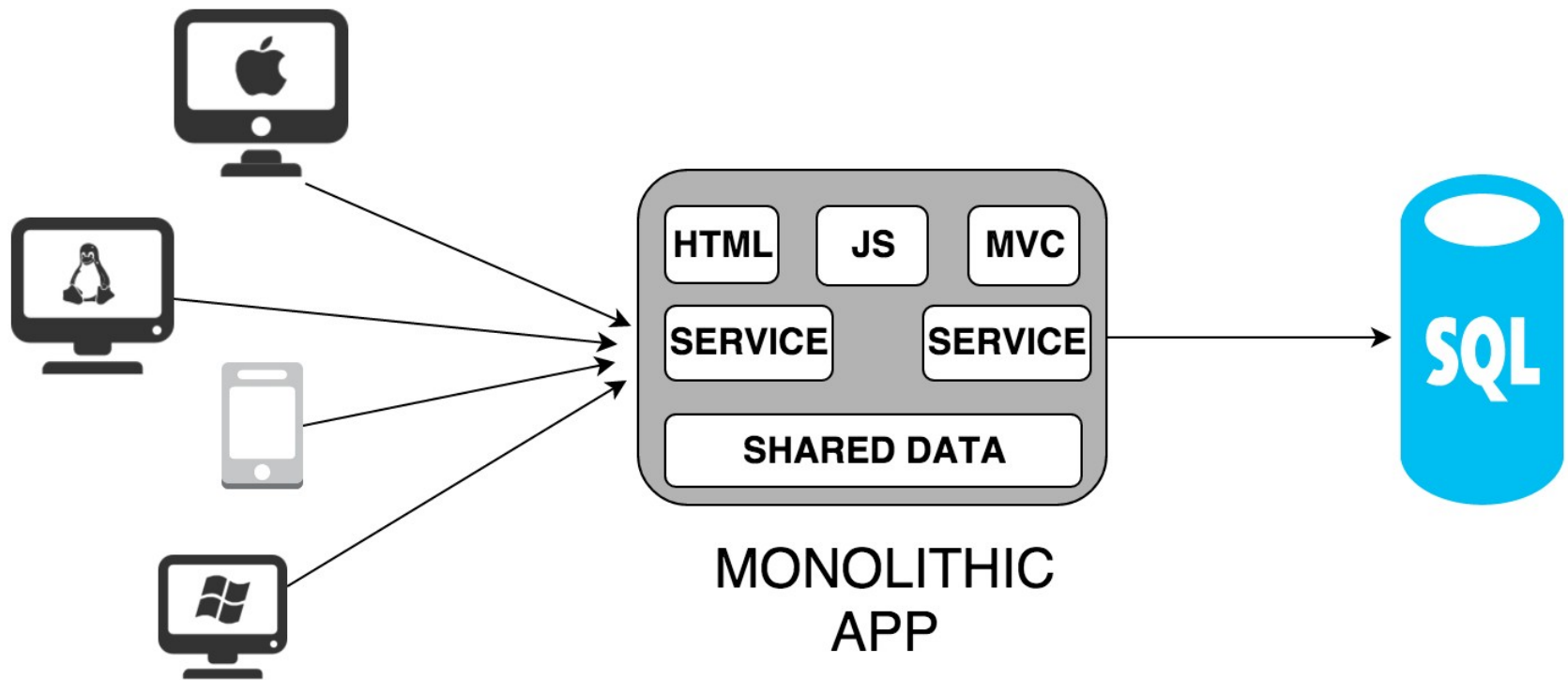
Monolithic vs Microservice

Architecture

Monolithic vs Microservice



Architecture (Monolithic)



Monolithic

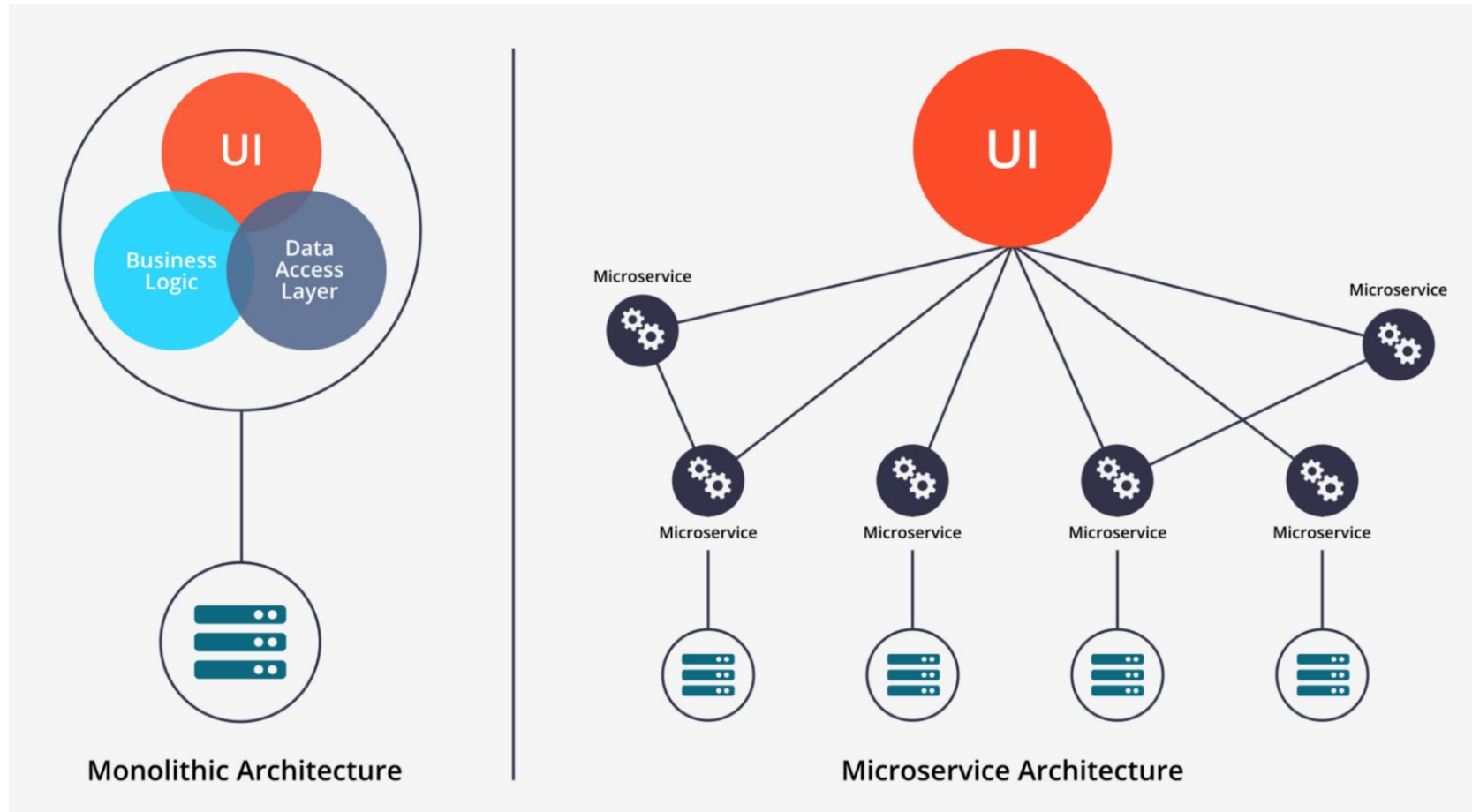
Pros

- Easy to develop
- Easy to deploy
- Easy to debug

Cons

- Difficult for Horizontal Scalling
- Tightly coupled
- Framework-centric

Architecture (Microservice)



Microservice

Pros

- Built for Horizontal scaling
- Independent
- Each service can be written in different programming language

Cons

- Difficult to develop
- Difficult to deploy
- Difficult to debug

Monolithic vs Microservice
Which one is Better?



Monolithic vs Microservice
Which one is Better?

It's depend

Monolithic vs Microservice

Best-cases

Monolithic

- Few users
- Single fighter
- Shared hosting

Microservice

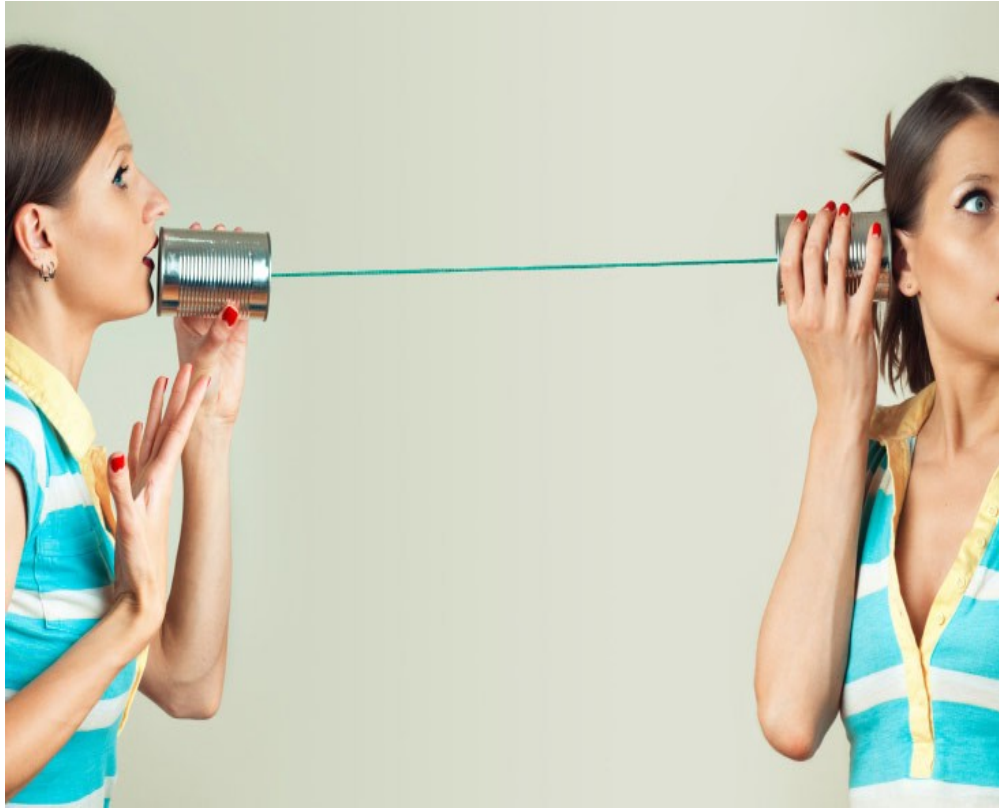
- A lot of users
- Teams of remote workers
- PaaS / IaaS

Communication

Pub/Sub vs Client/Server

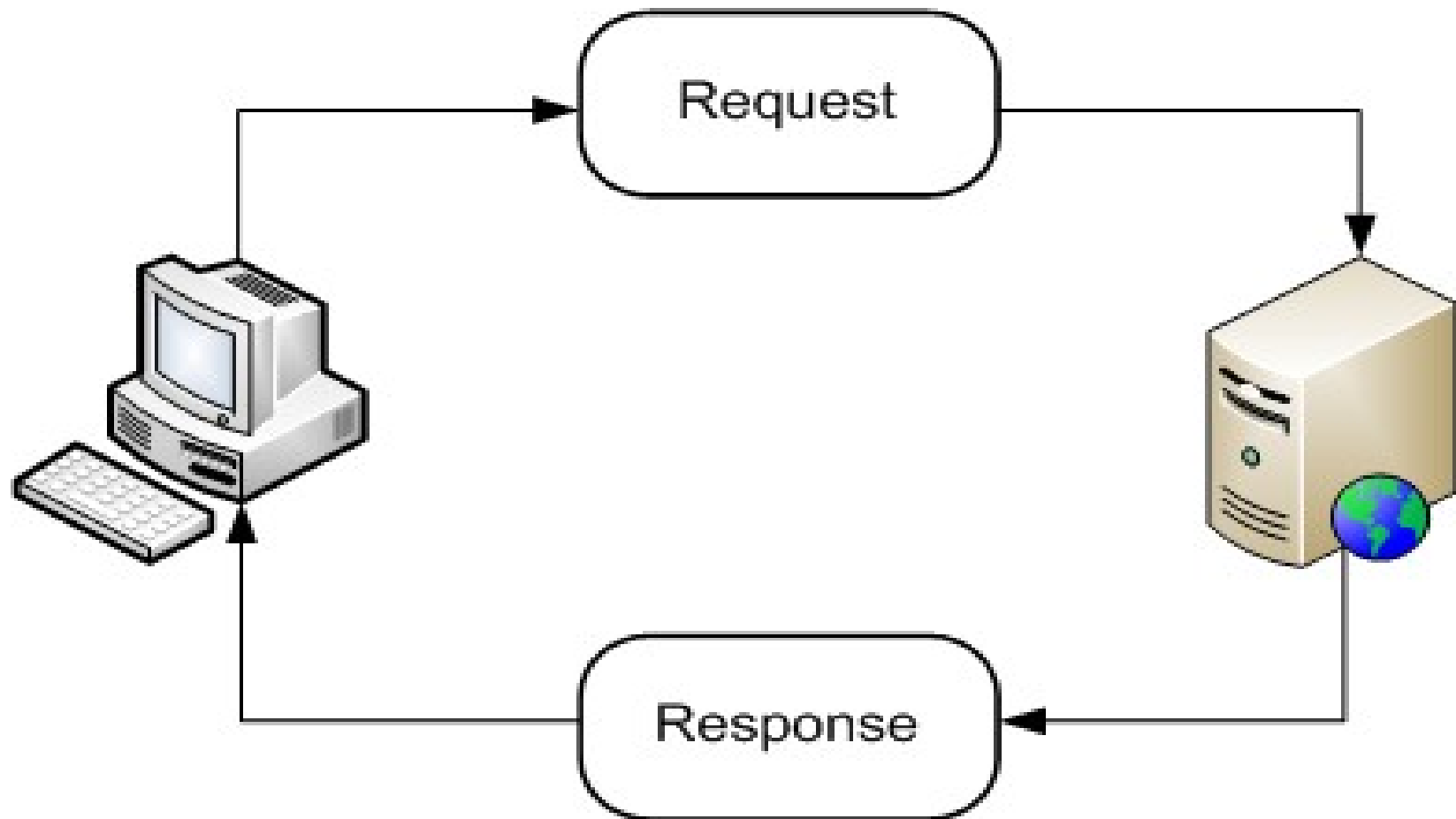
Communication

Req/Res vs Pub/Sub



Req/Res

Request/Response



Req/Res



```
curl http://localhost:3000
```

Client

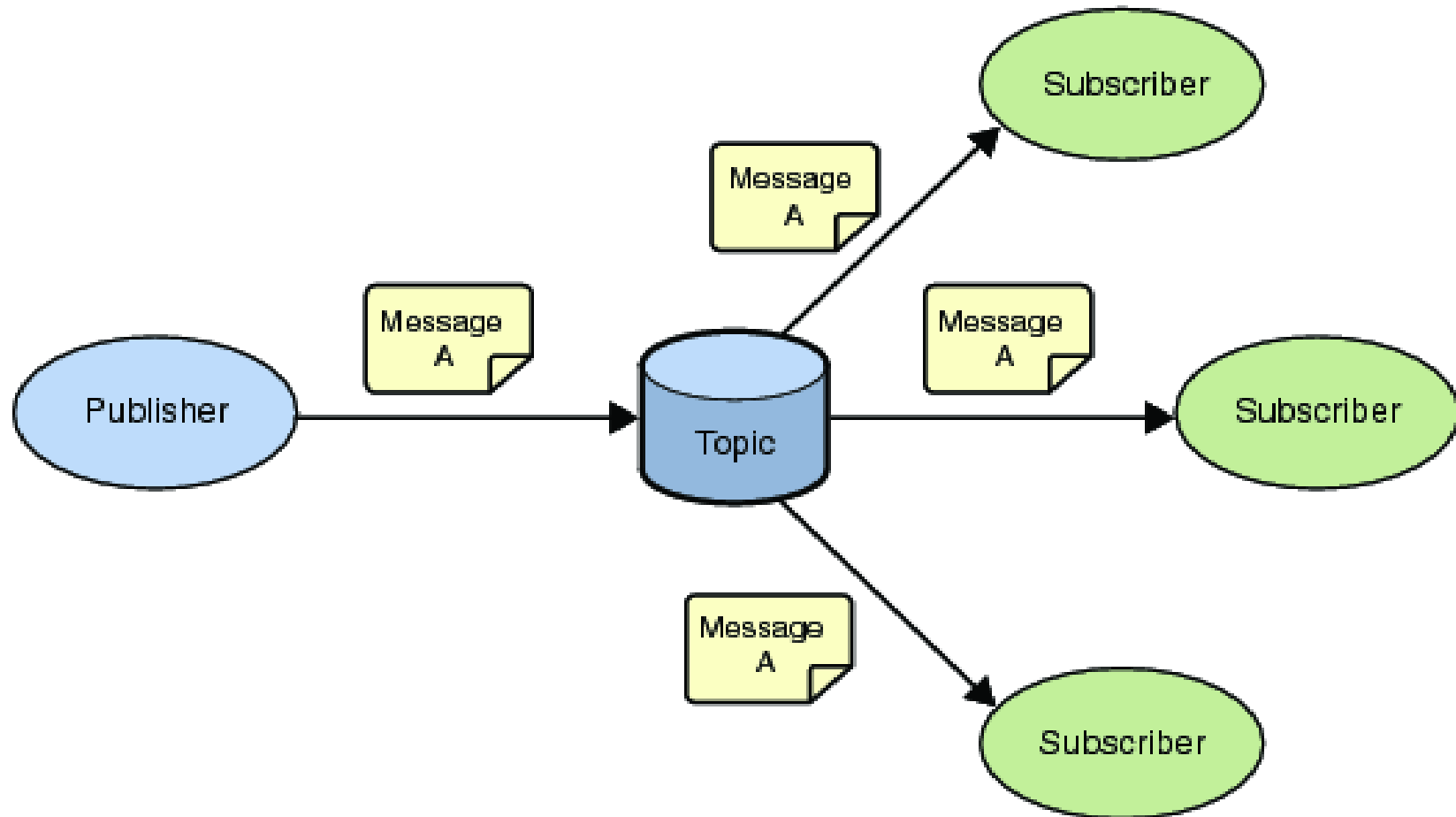


```
const express = require('express');  
const app = express();  
  
app.get('/', function(req, res){  
  res.send("Hello world!");  
});  
  
app.listen(3000);
```

Server

Pub/Sub

Publish/Subscribe



Pub/Sub



```
const NATS = require('nats');  
const nats = NATS.connect();  
  
// Simple Publisher  
nats.publish('foo', 'Hello World!');
```

Publisher



```
const NATS = require('nats');  
const nats = NATS.connect();  
  
// Simple Subscriber  
nats.subscribe('foo', function(msg) {  
  console.log('Received a message: ' + msg);  
});
```

Subscribers



```
const NATS = require('nats');  
const nats = NATS.connect();  
  
// Another Simple Subscriber  
nats.subscribe('foo', function(msg) {  
  console.log('Got: ' + msg);  
});
```

Req/Res vs Pub/Sub

Best-cases

Req/Res

- Immediate response
- Tight coupled services
- Single Listener

Pub/Sub

- No response needed
- Independent services
- Multiple Listeners

Moleculer

**Progressive microservices framework
for Node.js.**

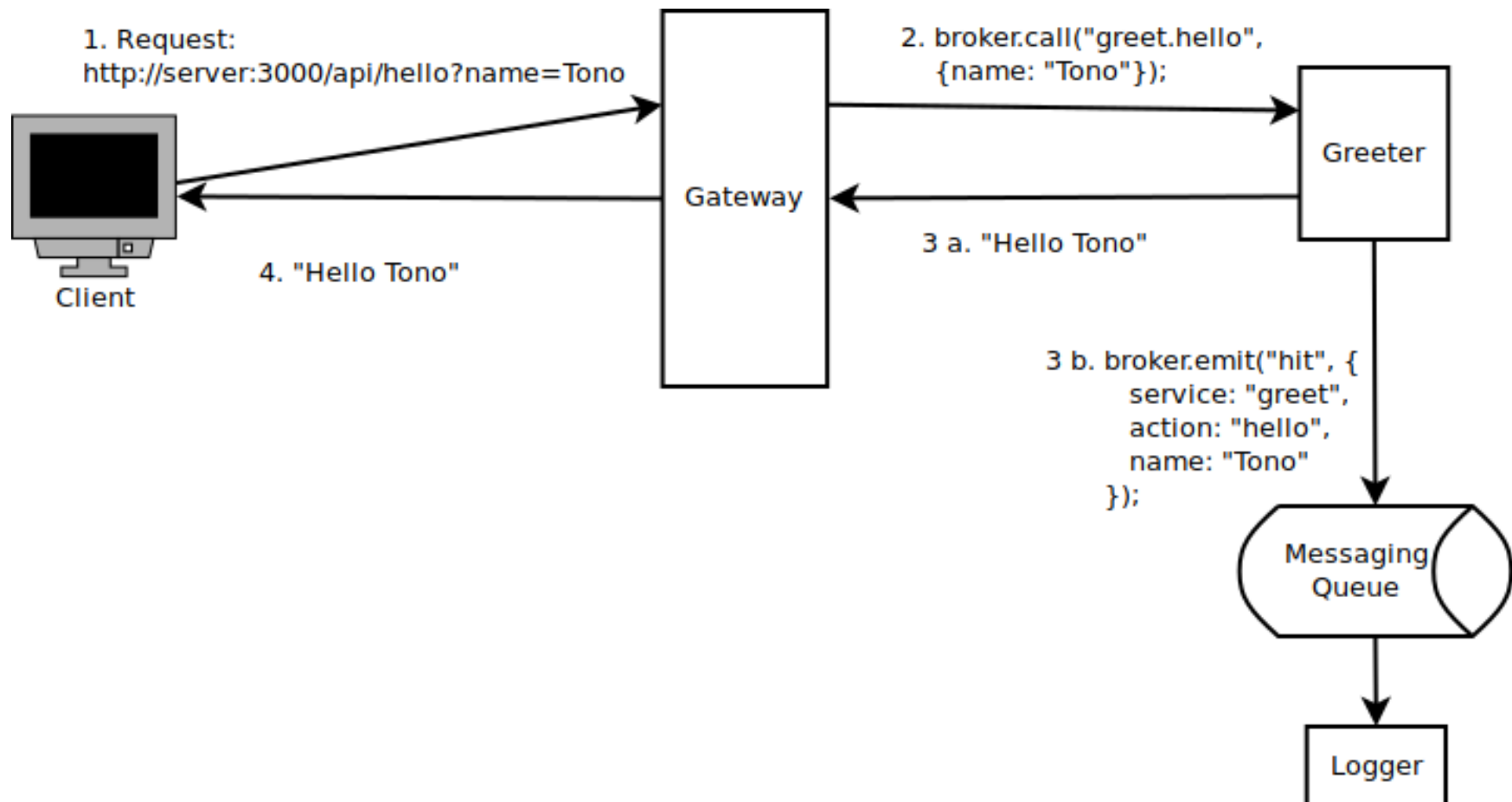
Molecular Service Broker

- Create Service
 - `broker.createService(serviceConfig);`
- Start
 - `broker.start();`
- Req/Res
 - `await broker.call("service.action", payload);`
- Publish
 - `broker.emit(event, payload);`


Let's Make It

Gateway → Greeter → Logger

The Blue Print



Gateway



```
const { ServiceBroker } = require("moleculer");
const ApiService = require("moleculer-web");

const broker = new ServiceBroker({
  transporter: "nats://0.0.0.0:4222",
});

broker.createService({
  mixins: [ApiService],
  settings: {
    port: 3000,
  },
  name: "api",
  actions: {
    async hello(ctx) {
      return await broker.call("greet.hello", {name: ctx.params.name});
    }
  }
});

broker.start();
```

Greeter




```
const { ServiceBroker } = require("moleculer");

const broker = new ServiceBroker({
  transporter: "nats://0.0.0.0:4222",
});

broker.createService({
  name: "greet",
  actions: {
    hello(ctx) {
      broker.emit("hit", {
        service: "greet",
        action: "hello",
        name: ctx.params.name,
      });
      return "Hello " + ctx.params.name;
    }
  }
});

broker.start();
```


Logger



```
const { ServiceBroker } = require("moleculer");

const broker = new ServiceBroker({
  transporter: "nats://0.0.0.0:4222",
});

broker.createService({
  name: "log",
  events: {
    "hit": {
      handler(payload) {
        console.log(payload);
      }
    }
  }
});

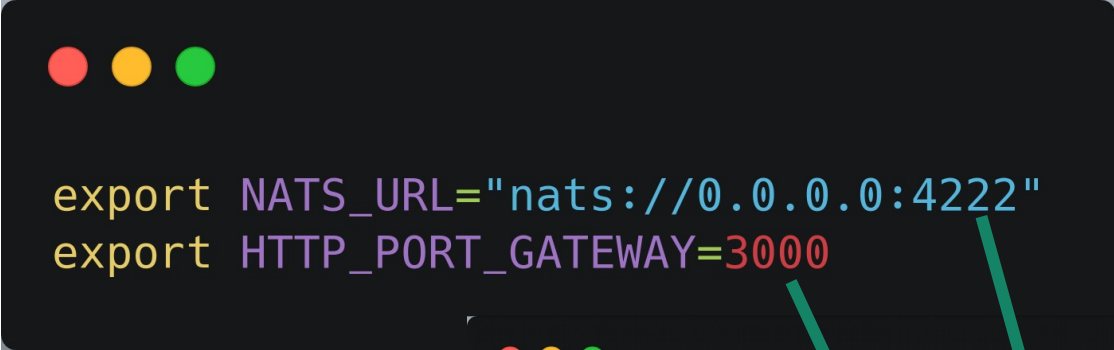
broker.start();
```

It's Done !!!

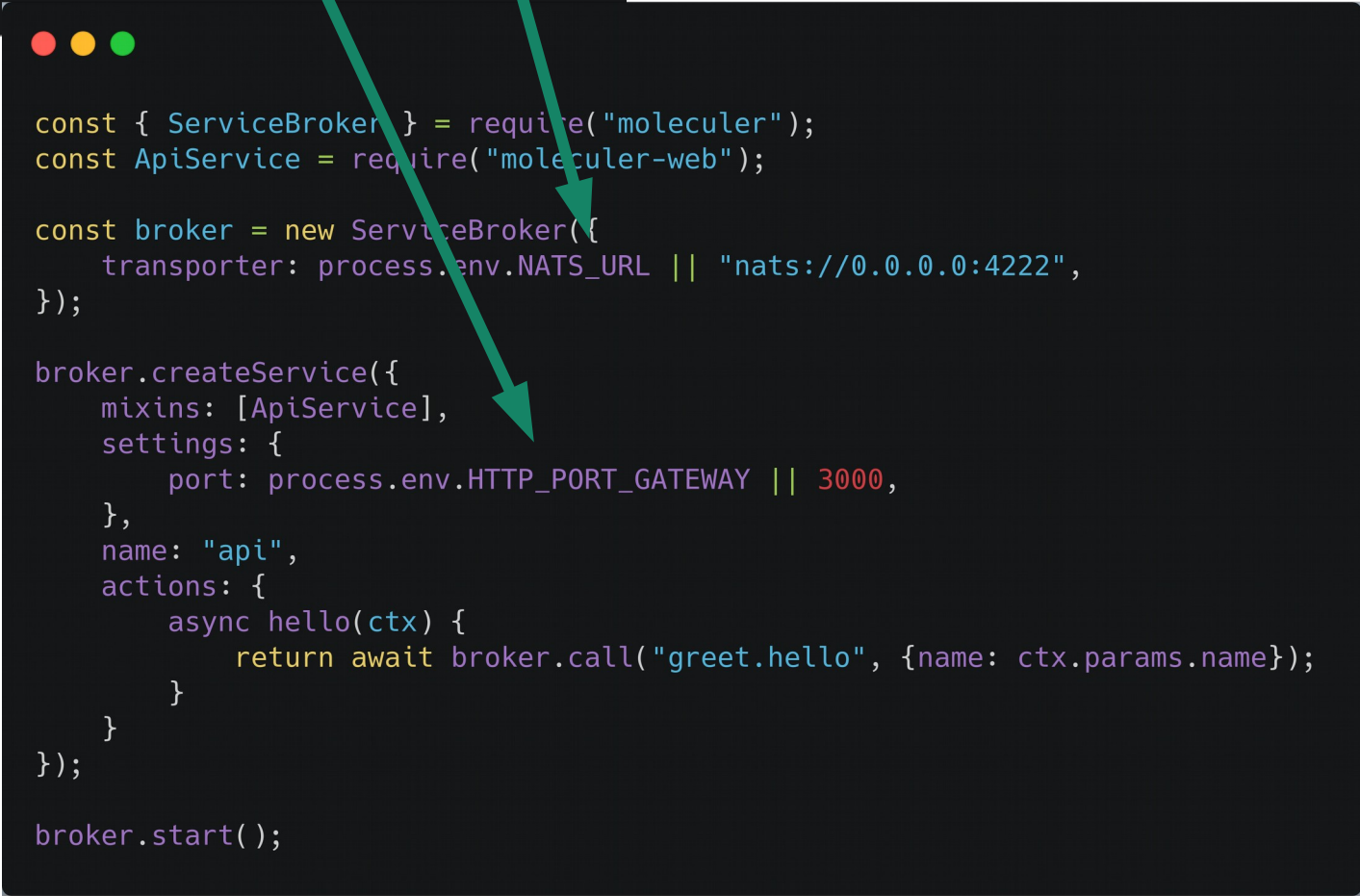
Bonus

Environment + Docker

Environment




```
export NATS_URL="nats://0.0.0.0:4222"  
export HTTP_PORT_GATEWAY=3000
```



```
const { ServiceBroker } = require("moleculer");  
const ApiService = require("moleculer-web");  
  
const broker = new ServiceBroker({  
  transporter: process.env.NATS_URL || "nats://0.0.0.0:4222",  
});  
  
broker.createService({  
  mixins: [ApiService],  
  settings: {  
    port: process.env.HTTP_PORT_GATEWAY || 3000,  
  },  
  name: "api",  
  actions: {  
    async hello(ctx) {  
      return await broker.call("greet.hello", {name: ctx.params.name});  
    }  
  }  
});  
  
broker.start();
```

Docker




```
FROM node:lts
```

```
WORKDIR /usr/src/app
```

```
COPY . .
```

```
RUN npm install
```

```
CMD [ "node", "./gateway.js" ]
```



```
docker build -t my-molecular-gateway .
```

```
docker run --name my-molecular-gateway --net=host -e NATS_PORT="nats://0.0.0.0:4222"  
-e HTTP_PORT_GATEWAY=3000 -p 3000:3000 -d my-molecular-gateway
```

Conclusion

- Microservice is more complex than monolithic
- Two Common way to communicate between services:
 - Req/Res
 - Pub/Sub
- Progressive microservice framework (like moleculer) make things easier
- Moleculer was built for Node.js. Using golang / python / other languages is not going to be easy
- There is no silver bullet

Further Reading

- <https://moleculer.services/>
- <https://microservices.io/>
- <https://www.martinfowler.com/articles/microservices.html>

Special Thanks

Ziyad Bazed

(<https://www.facebook.com/ziyadbazed>) for correcting some miss-conception about micro-service scalability.