

Building software to run on the Cloud

# Anatomy of an Internet Service

HTTP

Clients and Servers

Requests and Responses

# HTTP Request/Response

Demo: `nc www.example.com 80`

# HTTP Request/Response

Demo:

nc `www.example.com` 80

host

port

# HTTP Request/Response

Demo:

*Request* → {

nc [www.example.com](http://www.example.com) 80

GET / HTTP/1.1

Host: [www.example.com](http://www.example.com)

# HTTP Request/Response

Demo:

*Request* → {

```
nc www.example.com 80
GET / HTTP/1.1
Host: www.example.com
```

```
HTTP/1.1 200 OK
Date: Wed, 10 Oct 2018 08:19:15 GMT
Content-Type: text/html
Content-Length: 1720
```

```
<!DOCTYPE html>
<html>
```

```
.
```

```
.
```

```
.
```

*Response*

# HTTP Request/Response

Demo:

*Request* → {

```
nc www.example.com 80
GET / HTTP/1.1
Host: www.example.com
```

*Headers* |

```
HTTP/1.1 200 OK
Date: Wed, 10 Oct 2018 08:19:15 GMT
Content-Type: text/html
Content-Length: 1720
```

```
<!DOCTYPE html>
<html>
.
.
.
```

*Response*

# HTTP Request/Response

Demo:

*Request* → {

```
nc www.example.com 80
GET / HTTP/1.1
Host: www.example.com
```

*Headers* |

```
HTTP/1.1 200 OK
Date: Wed, 10 Oct 2018 08:19:15 GMT
Content-Type: text/html
Content-Length: 1720
```

*Response* |  
*Body* |

```
<!DOCTYPE html>
<html>
.
.
.
```

*R Response*



# The Webserver

Where did the `www.example.com` HTTP response  
come from?

# The Webserver

Where did the `www.example.com` HTTP response come from?



# The local Webserver

Demo:

```
http -v http://localhost:8080
```

```
GET / HTTP/1.1
```

```
Host: localhost:8080
```

```
User-Agent: curl/7.58.0
```

```
Accept: */*
```

```
HTTP/1.1 200
```

```
Content-Type: text/html; charset=UTF-8
```

```
.
```

```
.
```

```
.
```

```
etc
```

# JSON

```
{  
  "name" : "Matthew",  
  "kids" : 2,  
  "likes" : ["dogs", "cheese", "sunshine"]  
}
```

# JSON

```
{  
  "id"      : 3,  
  "title"   : "buy carrots",  
  "completed" : false  
}
```

# Fetching JSON from the local Webserver

# Fetching JSON from the local Webserver

```
http localhost:8080/api/todos
```

```
HTTP/1.1 200
```

```
Content-Type: application/json;charset=UTF-8
```

```
Date: Wed, 10 Oct 2018 18:02:30 GMT
```

```
Transfer-Encoding: chunked
```

```
[]
```

# Fetching JSON from the local Webserver

```
http localhost:8080/api/todos
```

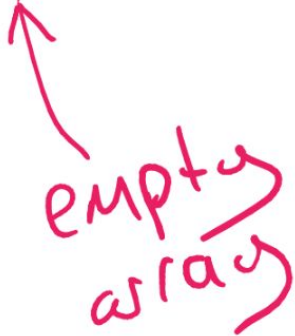
```
HTTP/1.1 200
```

```
Content-Type: application/json;charset=UTF-8
```

```
Date: Wed, 10 Oct 2018 18:02:30 GMT
```

```
Transfer-Encoding: chunked
```

```
[]
```



empty  
array



# Sending JSON to the local Webserver

# Sending JSON to the local Webserver

```
http POST http://localhost:8080/api/todos <<< '{"title": "buy carrots"}'
```

```
HTTP/1.1 200
```

```
Content-Type: application/json;charset=UTF-8
```

```
Date: Wed, 10 Oct 2018 18:17:13 GMT
```

```
Transfer-Encoding: chunked
```

```
{  
  "completed": false,  
  "id": 3,  
  "title": "buy carrots"  
}
```

# Sending JSON to the local Webserver

```
http POST http://localhost:8080/api/todos <<< '{"title": "buy carrots"}'
```

```
HTTP/1.1 200
```

```
Content-Type: application/json;charset=UTF-8
```

```
Date: Wed, 10 Oct 2018 18:17:13 GMT
```

```
Transfer-Encoding: chunked
```

```
{  
  "completed": false,  
  "id": 1,  
  "title": "buy carrots"  
}
```

"Method"  
or "Verb"

# Sending JSON to the local Webserver

```
http PUT http://localhost:8080/api/todos/1 <<< '{"title": "buy carrots",  
"completed":true}'
```

```
HTTP/1.1 200
```

```
Content-Type: application/json;charset=UTF-8
```

```
Date: Wed, 10 Oct 2018 18:18:39 GMT
```

```
Transfer-Encoding: chunked
```

```
{  
    "completed": true,  
    "id": 1,  
    "title": "buy carrots"  
}
```

# Sending JSON to the local Webserver

```
http PUT http://localhost:8080/api/todos/1 <<< '{"title": "buy carrots",  
"completed":true}'
```

HTTP/1.1 200

Content-Type: application/json;charset=UTF-8

Date: Wed, 10 Oct 2018 18:18:39 GMT

Transfer-Encoding: chunked

```
{  
  "completed": true,  
  "id": 1,  
  "title": "buy carrots"  
}
```



# A Web Application

Q: What do we GET from `http://localhost:8080/` ?

# A Web Application

Q: What do we GET from `http://localhost:8080/` ?

A: Some HTML

# A Web Application

```
<!doctype html>
<html>
<head>
  <title>UOB Todo App</title>
  <link rel="stylesheet" href="/static/css/main.css">
</head>
<body>
<div id="app"></div>
<script src="/static/js/app.js"></script>
</body>
</html>
```



# A Web Application



```
<!doctype html>
<html>
<head>
  <title>UOB Todo App</title>
  <link rel="stylesheet" href="/static/css/main.css">
</head>
<body>
<div id="app"></div>
<script src="/static/js/app.js"></script>
</body>
</html>
```

# A Web Application

```
<!doctype html>
<html>
<head>
  <title>UOB Todo App</title>
  <link rel="stylesheet" href="/static/css/main.css">
</head>
<body>
<div id="app"></div>
<script src="/static/js/app.js"></script>
</body>
</html>
```

HTML

stylesheet

Javascript

# JavaScript

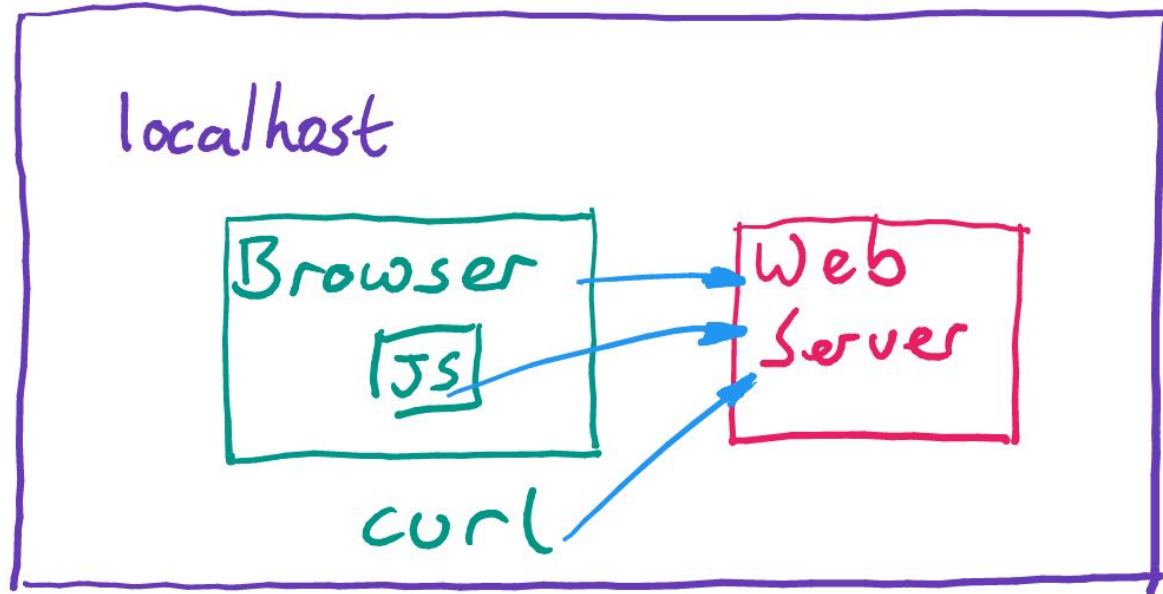
Q: JavaScript ?

# JavaScript

Q: JavaScript ?

- A programming language (not related to Java)
- Run by the browser
- In the browser, JavaScript code can:
  - Add/remove/change things on the page
  - Make more HTTP requests (to the same server)

# “Architecture”



# The Webserver Code



# The Webserver Code

Example:

```
@Controller
public class IndexController {

    @GetMapping("/")
    public String index() {
        return "index";
    }
}
```

# The Webserver Code

Example:

```
@Controller
public class IndexController {

    @GetMapping("/")
    public String index() {
        return "index";
    }
}
```

Because of the `@Controller` annotation, `IndexController` is a “Spring Bean” - ie it’s created and managed by Spring code



# The Webserver Code

Types of Spring Bean which can be used for responding to HTTP requests:

- @Controller
- @RequestMapping
- @RestController

# The Webserver Code

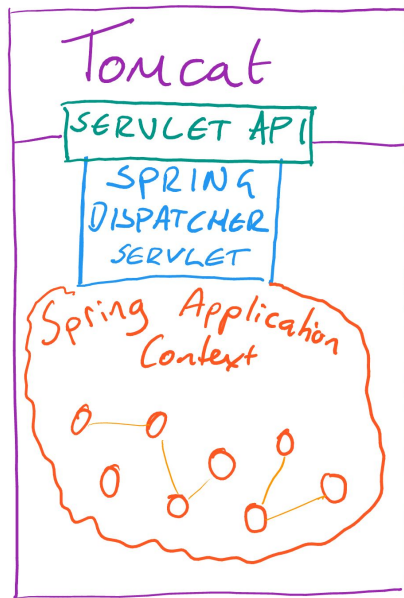
Types of Spring Bean which can be used for responding to HTTP requests:

@Controller  
@RequestMapping  
@RestController

... but HOW ??!

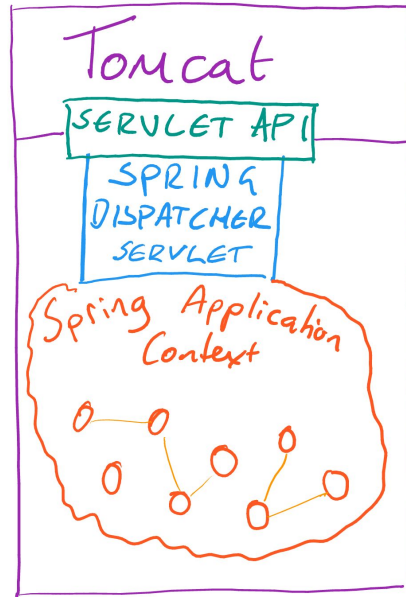
# The Webserver Code

Inside the Java app:



# The Webserver Code

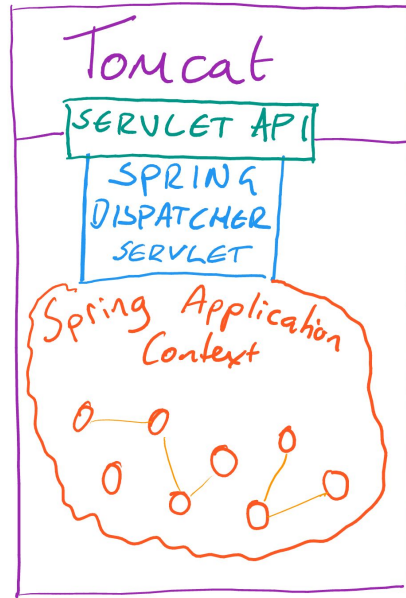
Inside the Java app:



READS AND WRITES  
ACTUAL HTTP

# The Webserver Code

Inside the Java app:

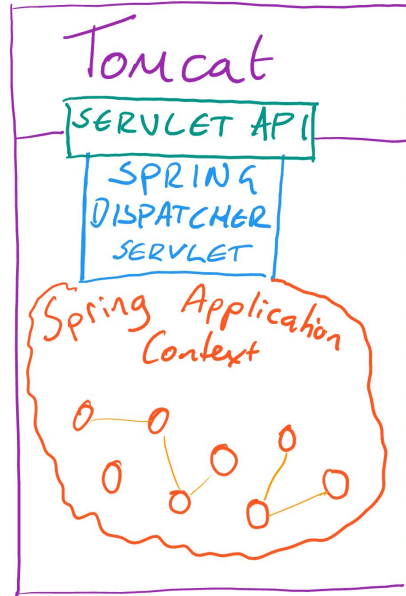


READS AND WRITES  
ACTUAL HTTP

JAVA STANDARD FOR  
WEB SERVICES

# The Webserver Code

Inside the Java app:



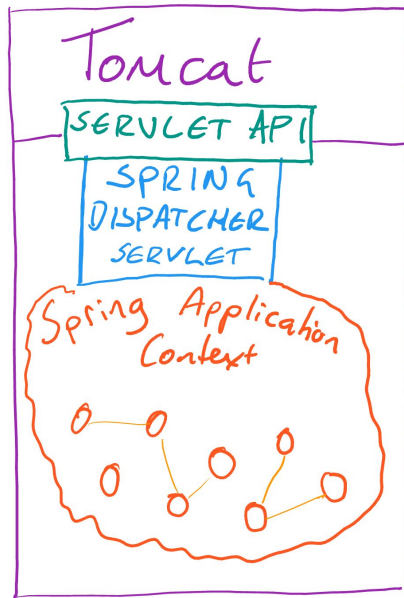
READS AND WRITES  
ACTUAL HTTP

JAVA STANDARD FOR  
WEB SERVICES

FINDS THE RIGHT "BEAN"  
AND CALLS A METHOD ON IT

# The Webserver Code

Inside the Java app:



READS AND WRITES  
ACTUAL HTTP

JAVA STANDARD FOR  
WEB SERVICES

FINDS THE RIGHT "BEAN"  
AND CALLS A METHOD ON IT

"BEANS" ARE JUST INSTANCES  
OF YOUR CLASSES WHICH  
SPRING IS MANAGING

# The Client Code



`vue.js`



# The Client Code

Vue files contain:

Templates

Bindings

Functions

(show some code)

# Calling Server code from the Client

```
axios.post('/api/todos', {"title": this.newTitle,})
```

```
@PostMapping()  
public TodoItem createTodo(@RequestBody TodoItem item) {  
    if (item.getTitle().equals("")) {  
        throw new BadRequestException("empty 'title'");  
    }  
    return todoSource.save(item);  
}
```

# Calling Server code from the Client

*This* →

```
axios.post('/api/todos', {"title": this.newTitle,})
```

```
@PostMapping()
```

```
public TodoItem createTodo(@RequestBody TodoItem item) {  
    if (item.getTitle().equals("")) {  
        throw new BadRequestException("empty 'title'");  
    }  
    return todoSource.save(item);  
}
```

# Calling Server code from the Client

*This* →

```
axios.post('/api/todos', {"title": this.newTitle,})
```

← *Calls This*

```
@PostMapping()
```

```
public TodoItem createTodo(@RequestBody TodoItem item) {  
    if (item.getTitle().equals("")) {  
        throw new BadRequestException("empty 'title'");  
    }  
    return todoSource.save(item);  
}
```

# Calling Server code from the Client

*This* →

```
axios.post('/api/todos', {"title": this.newTitle,})
```

← *Calls This*

```
@PostMapping()
```

```
public TodoItem createTodo(@RequestBody TodoItem item) {  
    if (item.getTitle().equals("")) {  
        throw new BadRequestException("empty 'title'");  
    }  
    return todoSource.save(item);  
}
```

*what's this??*

# Where is the data stored?

```
@Autowired  
public TodoController(TodoRepository todoSource){  
    this.todoSource = todoSource;  
}
```

# Where is the data stored?

```
@Autowired  
public TodoController(TodoRepository todoSource){  
    this.todoSource = todoSource;  
}
```

“Dependency Injection”

# Where is the data stored?

In the `TodoRepository`



# Where is the data stored?

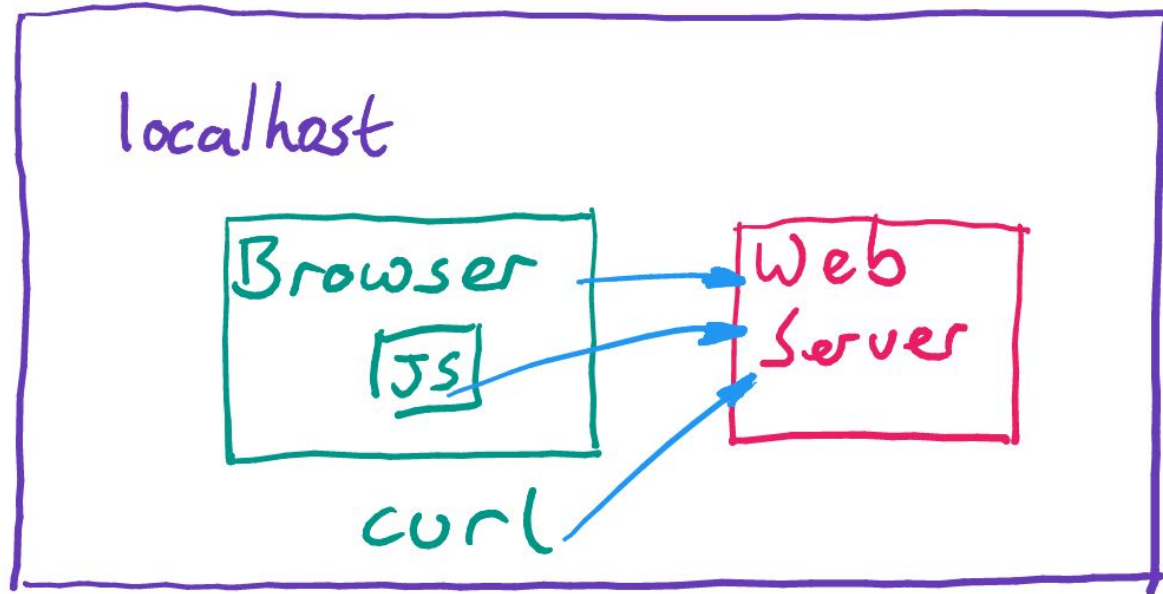
*memory*  
In the ~~TodoRepository~~

# Where is the data stored?

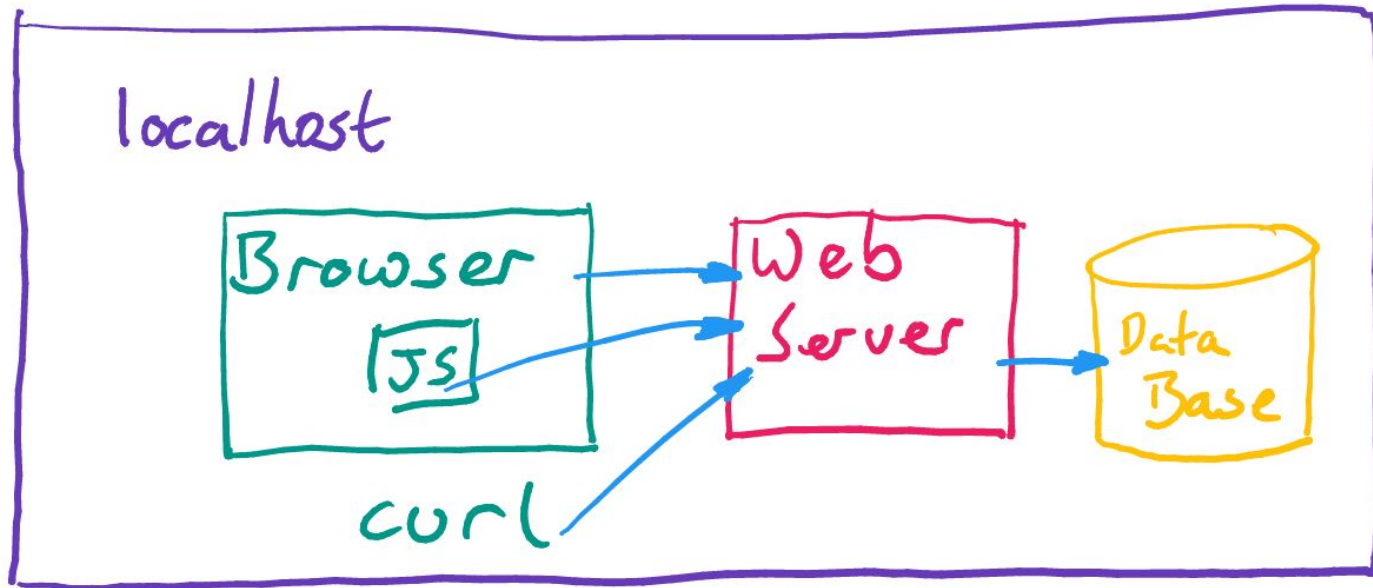
~~In the ~~TodoRepository~~~~ *memory*

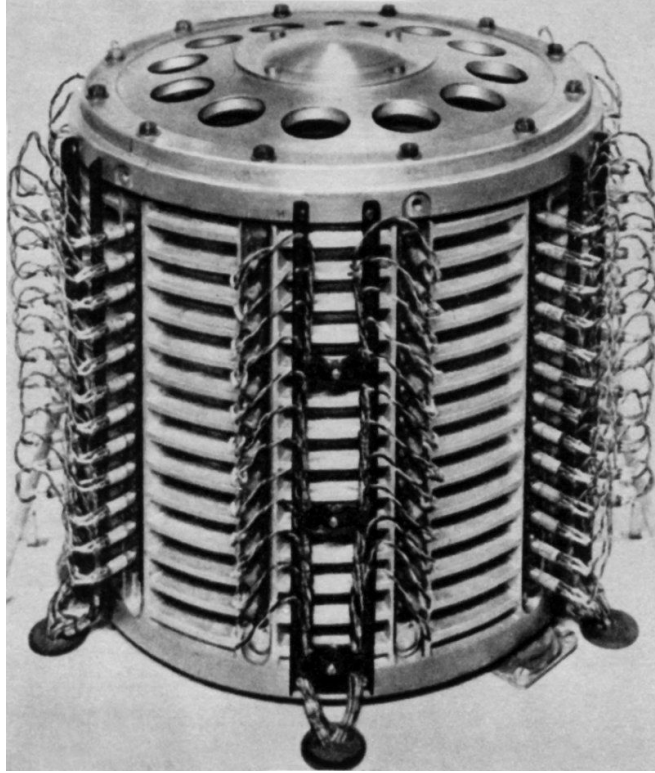


# “Architecture”



# “Architecture”







Who can use this app?

# Who can use this app?

Answer:

You, by using localhost



# Who can use this app?

Answer:

People on the same network as you, if they know your IP address.

# Who can use this app?

Answer:

People on the same network as you, if they know your IP address.

localhost → 127.0.0.1

# Who can use this app?

Answer:

People on the same network as you, if they know your IP address.

localhost → 127.0.0.1

192.168.9.43

# From the internet?

Probably not.

Your computer is not connected directly to the internet, but through a router.

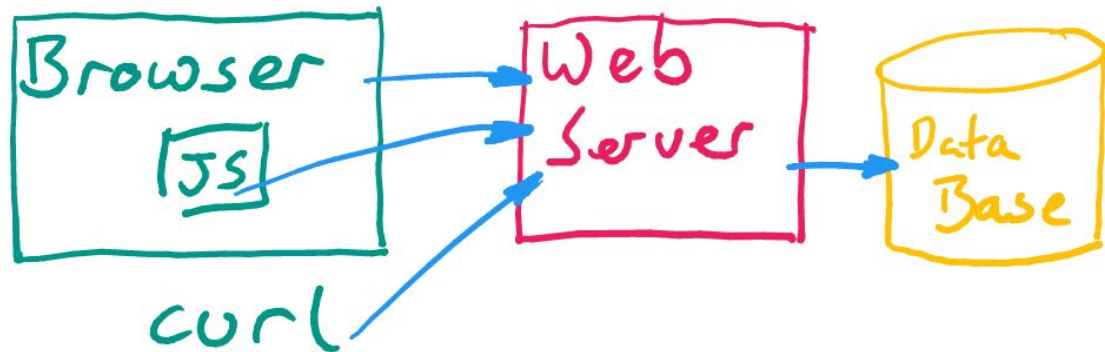
# From the internet?

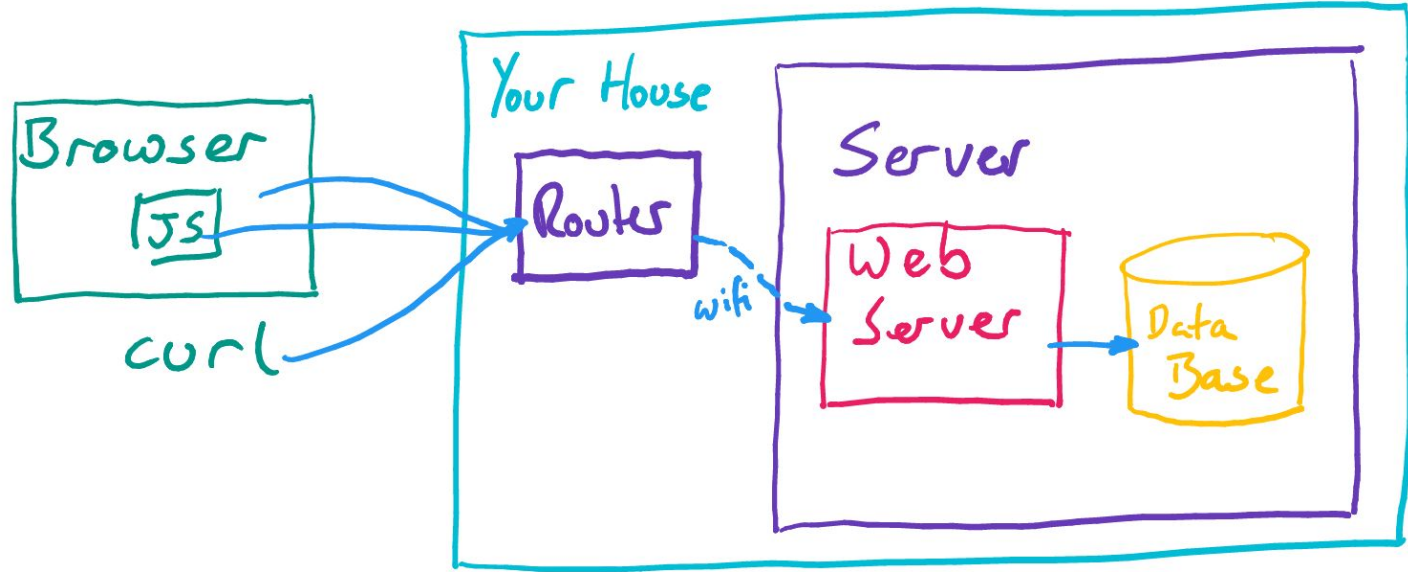
Probably not.

Your computer is not connected directly to the internet, but through a router.

Configure the router to forward connections to your computer

localhost





# How to measure reliability?



# How to measure reliability?

Simplest:

What % of the time is the web application useable?

# How to measure reliability?

Simplest:

What % of the time is the web application useable?

99%

# How to measure reliability?

Simplest:

What % of the time is the web application useable?

99% => unavailable for 7 hours/month

# How to measure reliability?

Simplest:

What % of the time is the web application useable?

99% => unavailable for 7 hours/month

99.9% => unavailable for 43 minutes/month

# How to measure reliability?

Simplest:

What % of the time is the web application useable?

99% => unavailable for 7 hours/month

99.9% => unavailable for 43 minutes/month

99.99% => unavailable for 4 minutes/month

# How to measure reliability?

Simplest:

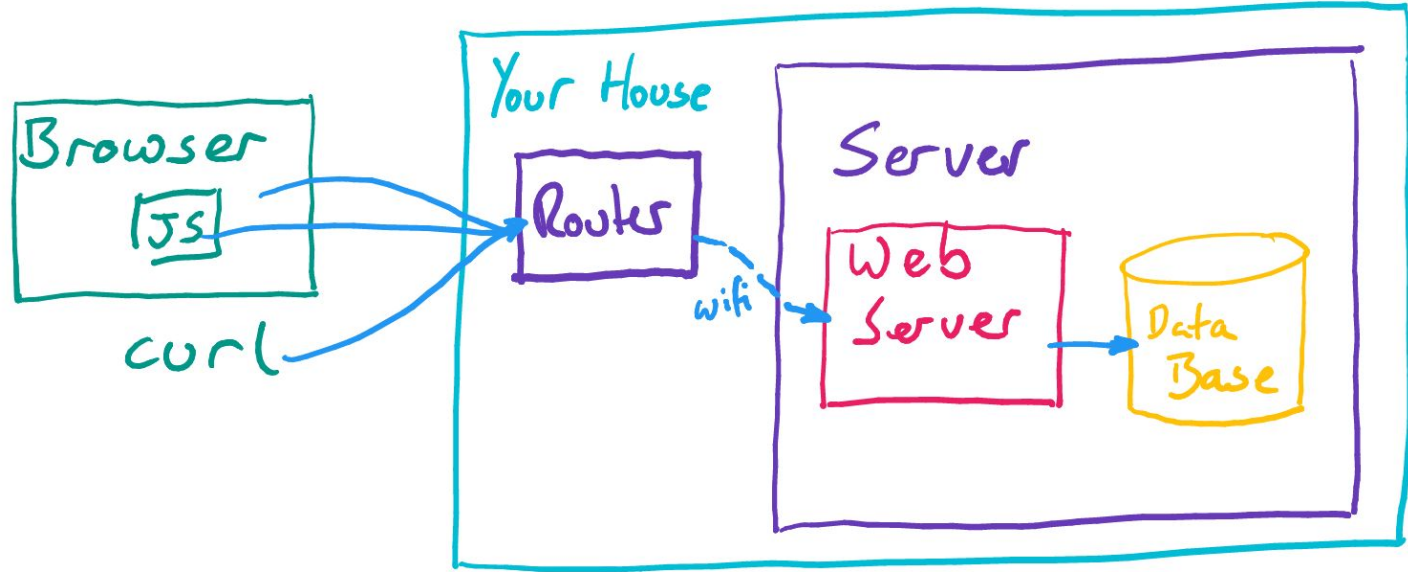
What % of the time is the web application useable?

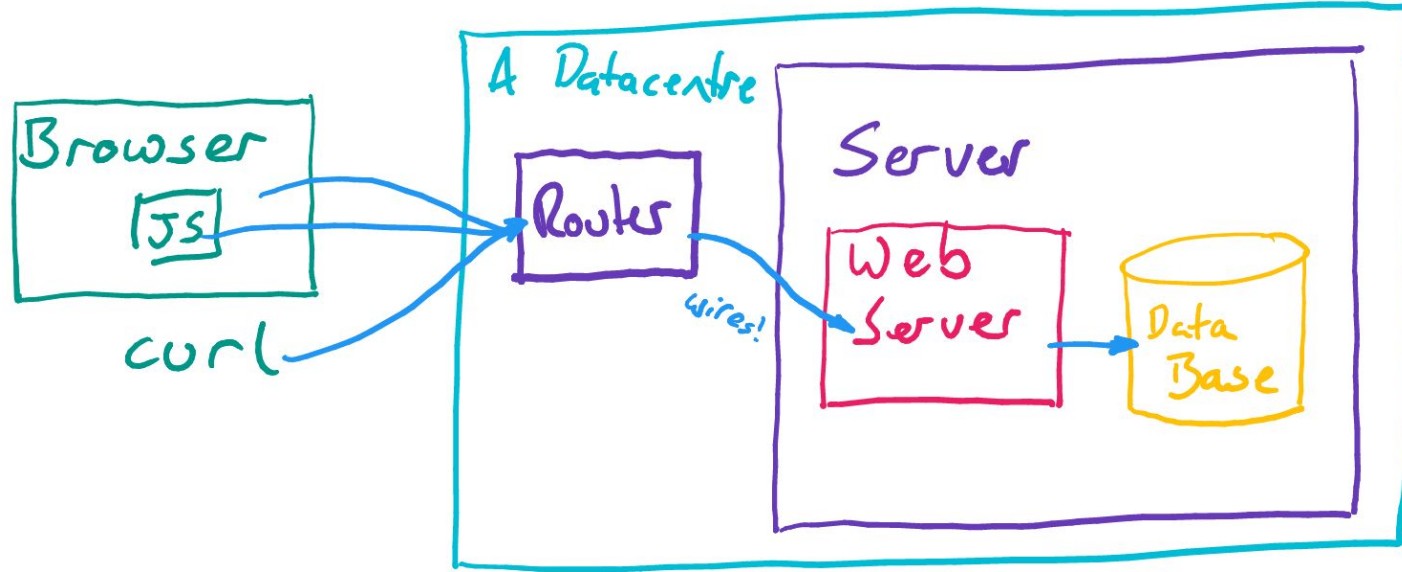
99% => unavailable for 7 hours/month

99.9% => unavailable for 43 minutes/month

99.99% => unavailable for 4 minutes/month

99.999% => unavailable for 26 seconds/month







# We have solved some problems

- Temperature, power, network all very reliable
- IP addresses are fixed

# We still have some problems

- We had to buy a computer and put it in the DC
- Hardware failure.
- Scaling.
- Updating your OS.
- Updating your application.

# We still have some problems

- We had to buy a computer and put it in the DC
- Hardware failure.
- Scaling.
- Updating your OS.
- Updating your application.

... solution?



There is no



...it's just someone else's computers