HTTP is your architecture

spot-media.de

Kore Nordmann <kore@qafoo.com>
Tobias Schlitt <toby@qafoo.com>

September 27, 2010

About us

- Long time FLOSS developers
- Open source enthusiast
- Authors, conference speakers
- Recently founded Qafoo GmbH - passion for software quality
- PMC members of Apache Zeta Components

Architecture

LCoDC$SS

Architecture

LCoDC$SS

- Who heard of this term before?

Architecture

LCoDC$SS

- Who heard of this term before?
  - This **is** HTTP. [Fie00]

Architecture

LCoDC$SS

Architecture

Layered CoDC$SS

Architecture

Layered Code on Demand C$SS

Architecture

Layered Code on Demand Client $S Server

Architecture

Layered Code on Demand Client Cached S

Server

Architecture

Layered Code on Demand Client Cached Stateless Server

Outline

HTTP

Layered

Conclusion

HTTP methods

- Well known methods
  - GET
  - POST

HTTP methods

- Well known methods
  - GET
  - POST
- Less known / used methods
  - PUT
  - DELETE

HTTP methods

- Well known methods
  - GET
  - POST
- Less known / used methods
  - PUT
  - DELETE
- Mostly unknown methods
  - HEAD
  - OPTIONS
  - TRACE
  - CONNECT
  - WebDAV
    - MKCOL
    - PROPSET
    - PROPGET
  - Use any you want...

contact@qafoo.com
http://talks.qafoo.com/

"Safe" HTTP methods

- "[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval." [RF99]
- . . . so it is "safe" for spiders to call them.

"Safe" HTTP methods

- "[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval." [RF99]
- . . . so it is "safe" for spiders to call them.
  - Search forms should use `method="GET"`

"Safe" HTTP methods

- "[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval." [RF99]
- ... so it is "safe" for spiders to call them.
  - Search forms should use `method="GET"`
- Since nothing is modified, the result can be cached.
  - Proxies can use that automatically
    - Varnish / Squid
    - Company application proxies

“Safe” HTTP methods

- “[..] GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval.” [RF99]
- . . . so it is “safe” for spiders to call them.
  - Search forms should use `method="GET"`
- Since nothing is modified, the result can be cached.
  - Proxies can use that automatically
    - Varnish / Squid
    - Company application proxies
  - **Layered** Code on Demand Client **Cached** Stateless Server

POST vs. PUT

- PUT creates or replaces a resource
  - "[..] requests that the enclosed entity be stored under th
    supplied Request-URI." [RF99]
  - "[..] refers to an already existing resource, the enclo
    SHOULD be considered as a modified version [..]" [RF9
  - Examples
    - Updating a users account data using `PUT /users/42`
    - Creation of resources with known identifiers
- ends to an existing resource
  - is used to request that the origin server accept the entity
    closed in the request as **a new subordinate**[..]" [RF99]
  - Examples [RF99]
    - Annotation of existing resources
    - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
    - Extending a database through an append operation.

POST vs. PUT

- PUT creates or replaces a resource
- POST appends to an existing resource

POST vs. PUT

- PUT creates or replaces a resource
- POST appends to an existing resource
  - "[..]is used to request that the origin server accept the entity enclosed in the request as **a new subordinate**[..]" [RF99]

POST vs. PUT

- PUT creates or replaces a resource
- POST appends to an existing resource
  - “[..]is used to request that the origin server accept the entity enclosed in the request as **a new subordinate**[..]” [RF99]
  - Examples [RF99]
    - Annotation of existing resources
    - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
    - Extending a database through an append operation.

POST vs. PUT

- PUT creates or replaces a resource
  - “[..] requests that the enclosed entity be stored under the supplied Request-URI.” [RF99]
  - “[..] refers to an already existing resource, the enclosed [illegible] SHOULD be considered as a modified version [..]” [RF99]
  - Examples
    - Updating a users account data using `PUT /users/42`
    - Creation of resources with known identifiers
- POST appends to an existing resource
  - “[..]is used to request that the origin server accept the entity enclosed in the request as **a new subordinate**[..]” [RF99]
  - Examples [RF99]
    - Annotation of existing resources
    - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
    - Extending a database through an append operation.

POST vs. PUT

- PUT creates or replaces a resource
  - “[..] requests that the enclosed entity be stored under the supplied Request-URI.” [RF99]
  - “[..] refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version [..]” [RF99]
  - Examples
    - Updating a users account data using PUT /users/42
    - Creation of resources with known identifiers
- POST appends to an existing resource
  - “[..]is used to request that the origin server accept the entity enclosed in the request as **a new subordinate**[..]” [RF99]
  - Examples [RF99]
    - Annotation of existing resources
    - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
    - Extending a database through an append operation.

POST vs. PUT

- PUT creates or replaces a resource
  - "[..] requests that the enclosed entity be stored under the supplied Request-URI." [RF99]
  - "[..] refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version [..]" [RF99]
  - Examples
    - Updating a users account data using `PUT /users/42`
    - Creation of resources with known identifiers
- POST appends to an existing resource
  - "[..]is used to request that the origin server accept the entity enclosed in the request as **a new subordinate**[..]" [RF99]
  - Examples [RF99]
    - Annotation of existing resources
    - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
    - Extending a database through an append operation.

contact@qafoo.com
http://talks.qafoo.com/

Idempotent methods

$$f(x) = f(f(x))$$

Idempotent methods

$$f(x) = f(f(x))$$

- Everything but `POST` has to be idempotent
  - Executing the request again, should not change anything.

Idempotent methods

$$f(x) = f(f(x))$$

- Everything but `POST` has to be idempotent
  - Executing the request again, should not change anything.
  - This includes `PUT` and `DELETE`

Idempotent methods

$$f(x) = f(f(x))$$

- Everything but POST has to be idempotent
  - Executing the request again, should not change anything.
  - This includes PUT and DELETE
- Really useful to just resend request, if one failed due to network problems

potence is a useful property in all messaging systems

Idempotent methods

$$f(x) = f(f(x))$$

- Everything but `POST` has to be idempotent
  - Executing the request again, should not change anything.
  - This includes `PUT` and `DELETE`
- Really useful to just resend request, if one failed due to network problems
- Idempotence is a useful property in all messaging systems

HTTP method fail in PHP

- $_GET contains the request parameters
- $_POST contains the request body

HTTP method fail in PHP

- `$_GET` contains the request parameters
- `$_POST` contains the request body
- All HTTP requests may contain a body **and** request parameters
  - Yes, even `GET`

HTTP method fail in PHP

- `$_GET` contains the request parameters
- `$_POST` contains the request body
- All HTTP requests may contain a body **and** request parameters
  - Yes, even `GET`
- You may want to use something like `$request->parameters` and `$request->body`

Do you speak HTTP?

- “The methods GET and HEAD MUST be supported by all general-purpose servers.” [RF99]

Do you speak HTTP?

- “The methods GET and HEAD MUST be supported by all general-purpose servers.” [RF99]
- “All other methods are OPTIONAL;” [RF99]

Do you speak HTTP?

- “The methods GET and HEAD MUST be supported by all general-purpose servers.” [RF99]
- “All other methods are OPTIONAL;” [RF99]
- “however, if the above methods are implemented, they MUST be implemented with the same semantics as those specified in section 9.” [RF99]

Do you speak HTTP?

- “The methods GET and HEAD MUST be supported by all general-purpose servers.” [RF99]
- “All other methods are OPTIONAL;” [RF99]
- “however, if the above methods are implemented, they MUST be implemented with the same semantics as those specified in section 9.” [RF99]
    - Sorry, your website is not HTTP, if you are using POST for a search form.
    - And the search results are not bookmarkable or linkable...

Outline

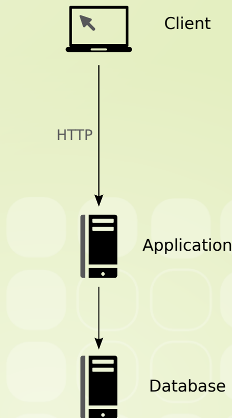
HTTP

Layered

Conclusion

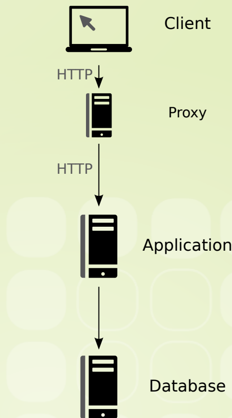
Layered architecture

- HTTP allows layered architectures
- But what is required to make this work?



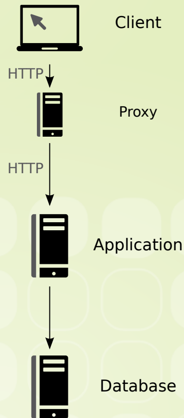
Layered architecture

- HTTP allows layered architectures



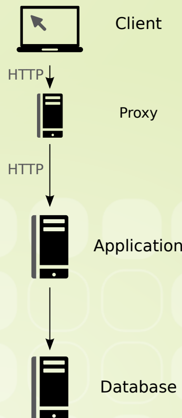
Layered architecture

- HTTP allows layered architectures
- But what is required to make this work?



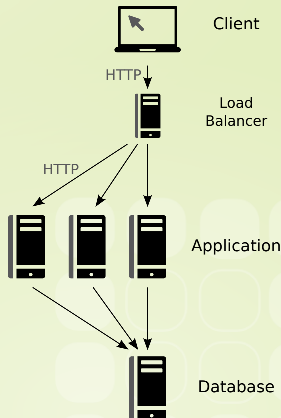
Layered architecture

- HTTP allows layered architectures
- But what is required to make this work?
  - Request semantic must be handled by the proxy



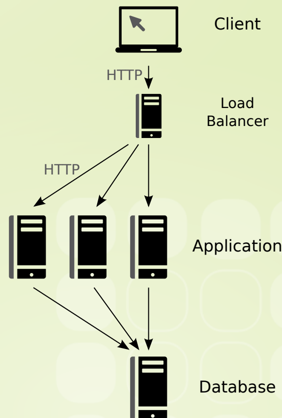
Layered architecture

- HTTP allows layered architectures
- But what is required to make this work?
  - Request semantic must be handled by the proxy



Layered architecture

- HTTP allows layered architectures
- But what is required to make this work?
    - Request semantic must be handled by the proxy
    - The server must be **stateless**



Stateless server

- No persistent connection
- Each request contains all information to be processed

Stateless server

- No persistent connection
- Each request contains all information to be processed
  - Cookies

Stateless server

- No persistent connection
- Each request contains all information to be processed
  - Cookies
- Servers can be exchanged transparently

Stateless server

- No persistent connection
- Each request contains all information to be processed
  - Cookies
- Servers can be exchanged transparently
  - Mind the sessions and static data

Drawbacks / Benefits

- Drawbacks
- Benefits

Drawbacks / Benefits

- Drawbacks
  - Users **do** have state – makes session handling complicated.
- Benefits

Drawbacks / Benefits

- Drawbacks
  - Users **do** have state – makes session handling complicated.
- Benefits
  - App servers do not scale beyond a single node
  - Session data can be hosted on a dedicated cluster
  - Caching servers do not matter
  - Can use plain random load balancing – massively reduces complexity of this layer
  - PHP scales, because it is build for this (Shared nothing architecture)

Drawbacks / Benefits

- Drawbacks
  - Users **do** have state – makes session handling complicated.
- Benefits
  - App servers do not scale beyond a single node
  - Session data can be hosted on a dedicated cluster

Drawbacks / Benefits

- Drawbacks
  - Users **do** have state – makes session handling complicated.
- Benefits
  - App servers do not scale beyond a single node
  - Session data can be hosted on a dedicated cluster
  - Failing servers do not matter

Drawbacks / Benefits

- Drawbacks
  - Users **do** have state – makes session handling complicated.
- Benefits
  - App servers do not scale beyond a single node
  - Session data can be hosted on a dedicated cluster
  - Failing servers do not matter
  - We can use plain random load balancing – massively reduces complexity of this layer
  - PHP scales, because it is build for this (Shared nothing architecture)

Drawbacks / Benefits

- Drawbacks
  - Users **do** have state – makes session handling complicated.
- Benefits
  - App servers do not scale beyond a single node
  - Session data can be hosted on a dedicated cluster
  - Failing servers do not matter
  - We can use plain random load balancing – massively reduces complexity of this layer
  - PHP scales, because it is build for this (Shared nothing architecture)

Embrace HTTP

- Use HTTP to communicate with backend services & subsystems
  - Webservices (REST, Soap, XMLRPC)
  - You can reuse your common infrastructure

Embrace HTTP

- Use HTTP to communicate with backend services & subsystems
  - Webservices (REST, Soap, XMLRPC)
  - You can reuse your common infrastructure
- Taking it to the next level
  - Use HTTP to communicate with your database (CouchDB)
  - Option to eliminate layers where appropriate

What is REST?

- What is REST actually?

What is REST?

- What is REST actually?
  - Describes services which follow the HTTP / LCoDC$SS

What is REST?

- What is REST actually?
  - Describes services which follow the HTTP / LCoDC$SS
  - Following the resources / concept character of URLs
  - … even respects the `Accept-*` HTTP headers

What is REST?

- What is REST actually?
  - Describes services which follow the HTTP / LCoDC$SS
  - Following the resources / concept character of URLs
  - Sometimes even respects the `Accept-*` HTTP headers

Outline

HTTP

Layered

Conclusion

Thanks for listening

- HTTP and PHP are build like this for a reason
  - Scalabilty
  - Fault tolerance
- More about us:
  - `http://qafoo.com`

Bibliography I

[Fie00] R. Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. thesis, University of California, Irvine, USA, 2000.

[RF99] et al. R. Fielding, *Hypertext transfer protocol – http/1.1*, http://tools.ietf.org/html/rfc2616, June 1999.