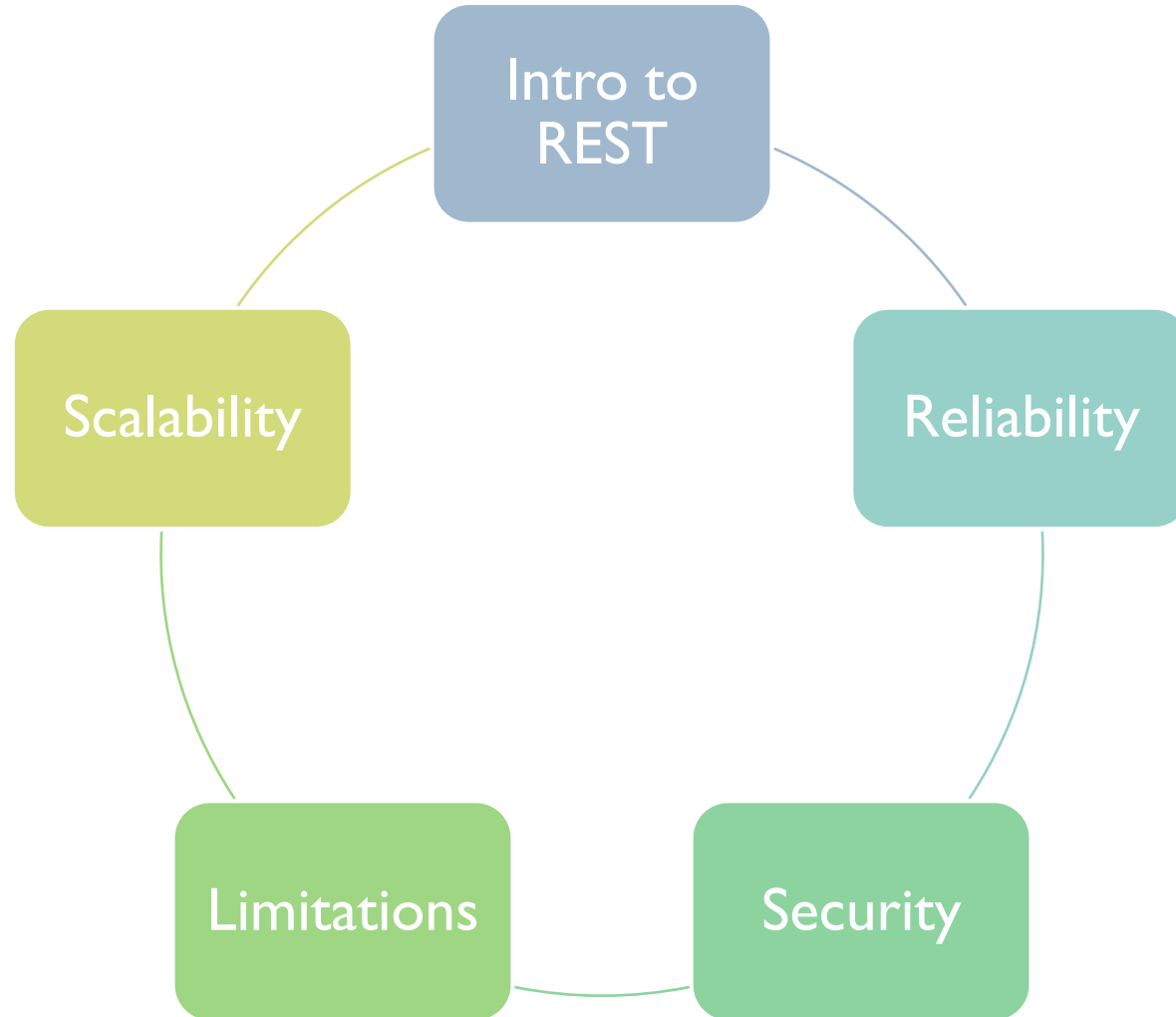


# Scalable, Reliable, and Secure RESTful services

A guide to building to HTTP based services

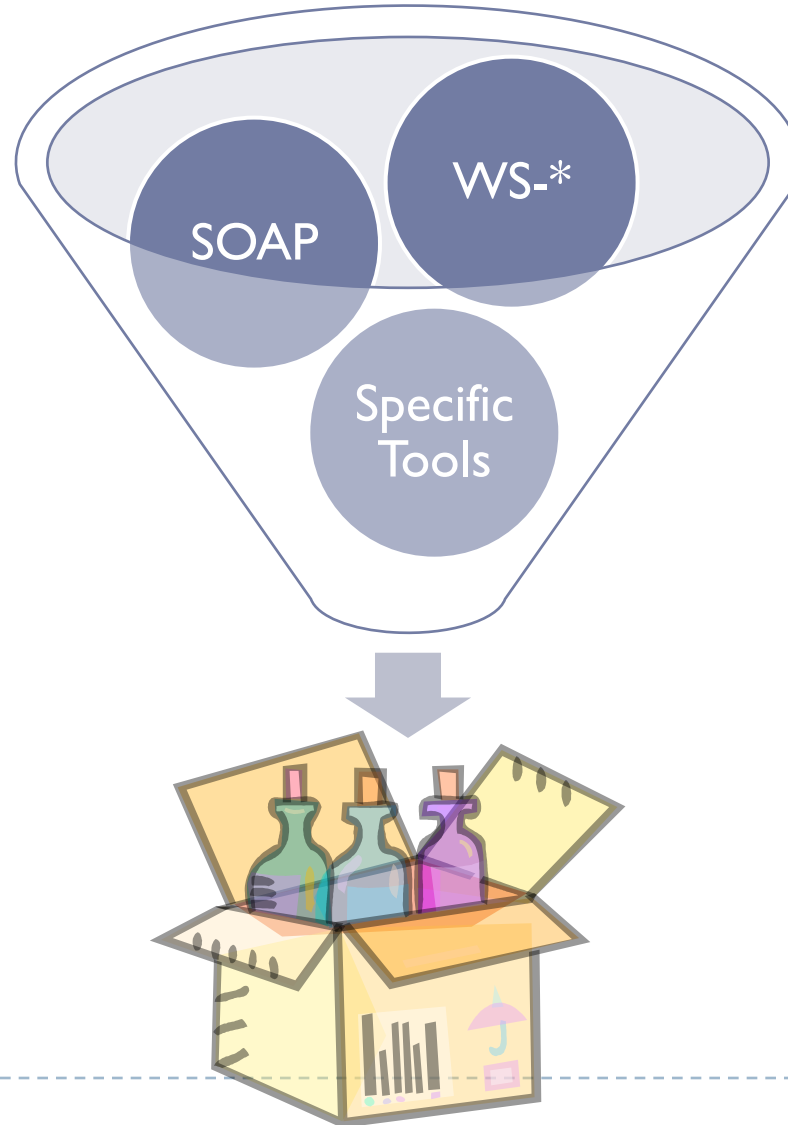
# Today's talk

---



# What this talk is NOT

---



# What is REST?

---

- ▶ Representation State Transfer
- ▶ Roy Fielding coined the term for his thesis which sought to provide the architectural basis for HTTP
- ▶ But what *is it*?





# Resources, resources, resources

---

- ▶ **Everything** is a resource
- ▶ Resources are addressable via **URIs**
- ▶ Resources are **self descriptive**
  - ▶ Typically through content types (“application/xml”) and sometimes the resource body (i.e. an XML QName)
- ▶ Resources are **stateless**
- ▶ Resources are manipulated via **verbs** and the **uniform interface**



# The Uniform Interface

---

**Uniform**

Get(URI)

Put(URI, Resource)

Delete(URI)

**Non Uniform**

getCustomer()

updateCustomer(Customer)

delete(customerId);

---

# Hypertext and linkability

---

- ▶ We don't want "keys", we want links!
- ▶ Resources are hypertext
  - ▶ Hypertext is just data with links to other resources
- ▶ Data model refers to other application states via links
- ▶ This is possible because of the uniform interface!  
No need to know different ways to get different types of entities!
- ▶ Client can simply navigate to different resources



# REST & HTTP

---

- ▶ REST defines the architectural style of HTTP
- ▶ We'll discuss RESTful principles in relation to HTTP specifically as we explore
  - ▶ Scalability
  - ▶ Reliability
  - ▶ Security



# Our Starting Point

---

**GET**

- Cacheable
- SAFE – no side effects

**POST**

- Unsafe operations, which can't be repeated

**PUT**

- Idempotent

**DELETE**

- Idempotent

**HEAD**

- SAFE – no side effects
  - No message body
- 



# Reliability through Idempotency

# Idempotent Operations

---

Same  
Request

yields



Same  
Result

---



# Some Basic Scenarios:

---

1. Getting resources
2. Deleting resources
3. Updating a resource
4. Creating a resource



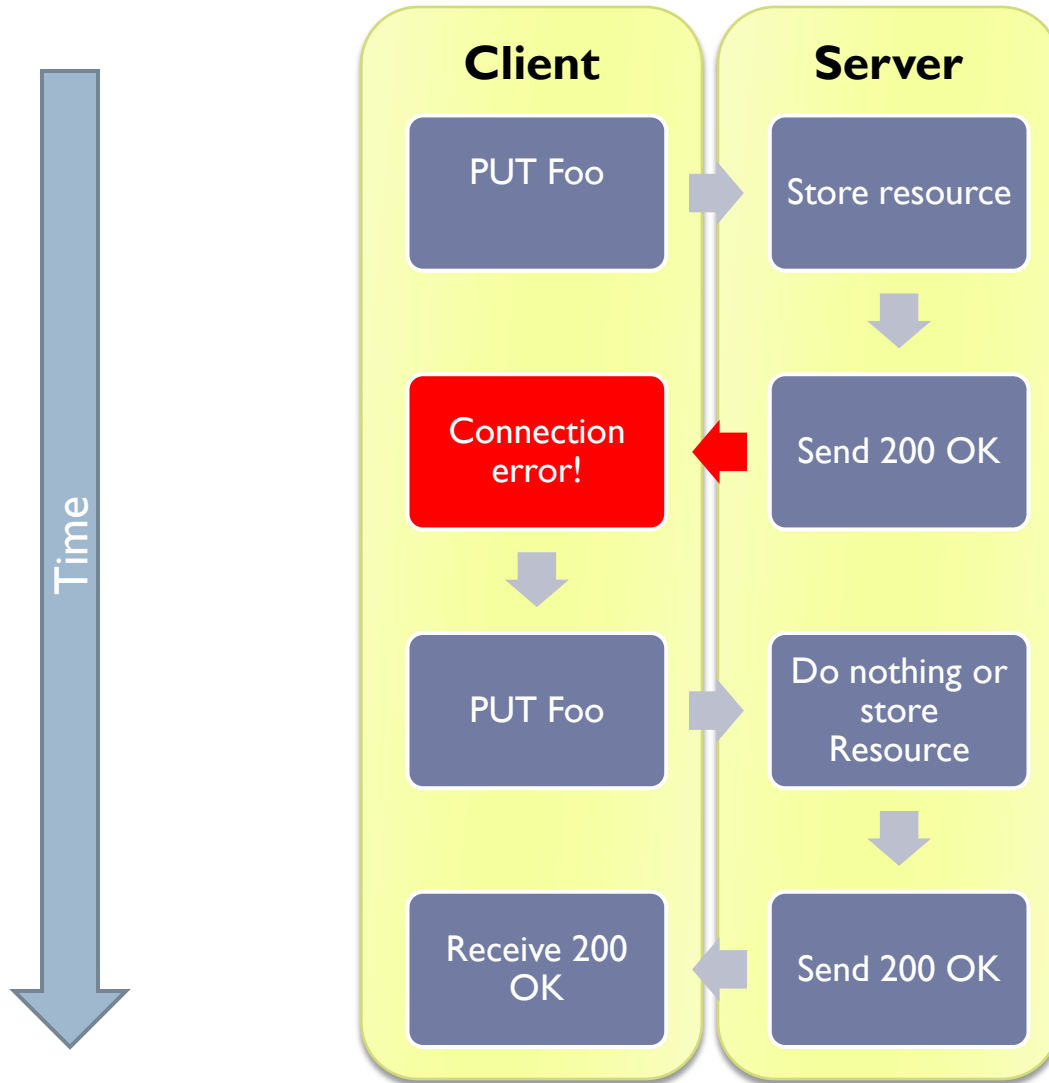
# Getting a resource

---

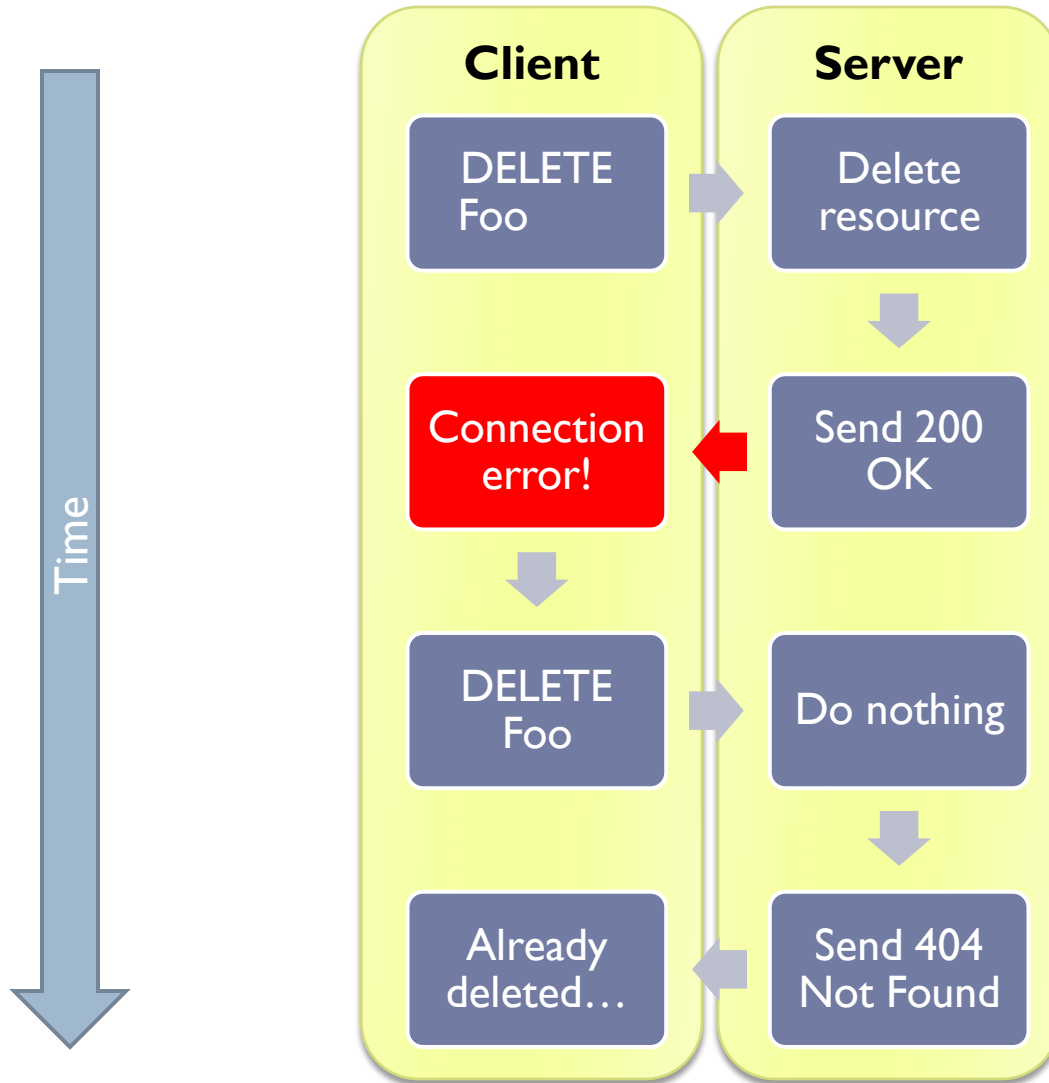
- ▶ **GET is SAFE**
- ▶ **If original GET fails, just try, try again**



# Updating a resource



# Deleting a resource



# Creating Resources

---

```
POST /entries
Host: acme.com
...
```

**Client**



```
HTTP/1.1 201 Created
Date: ...
Content-Length: 0
Location:
  http://acme.com/entries/1
...
```

**Server**

```
PUT /entries/1
Host: acme.com
Content-Type: ...
Content-Length: ...

Some data...
```

**Client**



```
HTTP/1.1 200 OK
...
```

**Server**

---



# Creating Resources

---

- ▶ IDs which are not used can be
  - ▶ Ignored
  - ▶ Expired
- ▶ Another option: have the client generate a unique ID and PUT to it straight away
  - ▶ They're liable to screw it up though



# Problem: Firewalls

---

- ▶ Many firewalls do not allow PUT, DELETE
- ▶ You might want to allow other ways of specifying a header:
  - ▶ Google: `X-HTTP-Method-Override: PUT`
  - ▶ Ruby: `?method=PUT`



# Scalability

ETags, Caching, Content-Types, URLs, and more

# Stateless client/server approach

---

- ▶ **All communication is stateless**
- ▶ **Session state is kept on the Client!**
  - ▶ Client is responsible for transitioning to new states
  - ▶ States are represented by URIs
- ▶ **Improves:**
  - ▶ Visibility
  - ▶ Reliability
  - ▶ Scalability



# ETag Header

---

- ▶ Resources may return an ETag header when it is accessed
- ▶ On subsequent retrieval of the resource, Client sends this ETag header back
- ▶ If the resource has not changed (i.e. the ETag is the same), an empty response with a 304 code is returned
- ▶ Reduces bandwidth/latency



# ETag Example

---

```
GET /feed.atom
Host: www.acme.com
...
```

**Client**



```
HTTP/1.1 200 OK
Date: ...
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
...
```

**Server**

```
GET /feed.atom
If-None-Match:
    "3e86-410-3596fbbc"
Host: www.acme.com
...
```

**Client**



```
HTTP/1.1 304 Not Modified
Date: ...
ETag: "3e86-410-3596fbbc"
Content-Length: 0...
```

**Server**

---



# LastModified Example

---

```
GET /feed.atom
Host: www.acme.com
...
```

```
GET /feed.atom
If-Modified-Since:
  Sat, 29 Oct 1994
  19:43:31 GMT
Host: www.acme.com
...
```

**Client**

```
HTTP/1.1 200 OK
Date: ...
Last-Modified: Sat, 29 Oct
  1994 19:43:31 GMT
Content-Length: 1040
Content-Type: text/html
...
```

```
HTTP/1.1 304 Not Modified
Date: ...
Last-Modified: Sat, 29 Oct
  1994 19:43:31 GMT
Content-Length: 0
```

**Server**

---

# Scalability through Caching

---

- ▶ A.k.a. “cache the hell out of it”
- ▶ Reduce latency, network traffic, and server load
- ▶ Types of cache:
  - ▶ Browser
  - ▶ Proxy
  - ▶ Gateway



# How Caching Works

---

- ▶ **A resource is eligible for caching if:**
  - ▶ The HTTP response headers don't say not to cache it
  - ▶ The response is not authenticated or secure
  - ▶ No ETag or LastModified header is present
  - ▶ The cache representation is fresh
- ▶ **From: [http://www.mnot.net/cache\\_docs/](http://www.mnot.net/cache_docs/)**



# Is your cache fresh?

---

- ▶ **Yes, if:**
  - ▶ The expiry time has not been exceeded
  - ▶ The representation was `LastModified` a relatively long time ago
- ▶ **If its stale, the remote server will be asked to *validate* if the representation is still fresh**



# Scalability through URLs and Content-Types

---

- ▶ Information about where the request is destined is held outside the message:
  - ▶ Content-Type
    - ▶ application/purchase-order+xml
    - ▶ image/jpeg
  - ▶ URL
  - ▶ Other headers
- ▶ Allows easy routing to the appropriate server with little overhead



# Transactions

---

- ▶ The web is **NOT** designed for transactions
  - ▶ Client is responsible for committing/rolling back transactions, and client may not fulfill responsibilities
  - ▶ Transactions can take too long over the web and tie up important resources
- ▶ Much better **IMO** to build in confirmation/cancellation into your application
- ▶ This requires application specific means for compensation
- ▶ See the paper: *Life Beyond Transactions* by Pat Helland



# So you really want transactions...

---

- ▶ People sometimes use HTTP for transactions
- ▶ Notable example: SVN
- ▶ It is possible to model a resource as a transaction
  - ▶ POST – create a new transaction
  - ▶ PUT – send “commit” state to transaction
  - ▶ DELETE – rollback the transaction





Security

# Question #1

---

- ▶ **What are your goals & requirements?**
  - ▶ Authentication?
  - ▶ Authorization?
  - ▶ Privacy?
  - ▶ Integrity?
  - ▶ Openness?
  - ▶ Eliminate hurdles for users?



# Tools at our disposal

---

- ▶ HTTP Authentication
- ▶ SSL
- ▶ XML Signature & Encryption
- ▶ Others:
  - ▶ SAML, Cardspace, OpenID...



# HTTP Authentication Basics

---

- ▶ **Basic Authentication**

- ▶ Username & Password passed in plain text

- ▶ **Digest**

- ▶ MD5 hash of username & password is created

- ▶ **Sent with every request**

- ▶ Remember – *statelessness?*



# SSL and Public Key Cryptography

---

- ▶ **SSL/TLS defines a process to encrypt/secure transports**

Negotiate an appropriate encryption algorithm



Exchange public keys and certificates



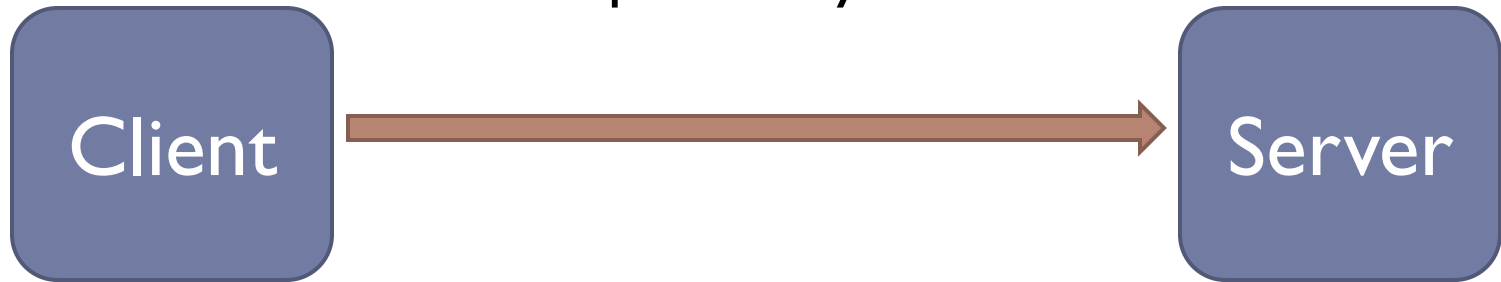
Negotiate a “common secret” which allows the connection to use symmetric cryptography



# How SSL works

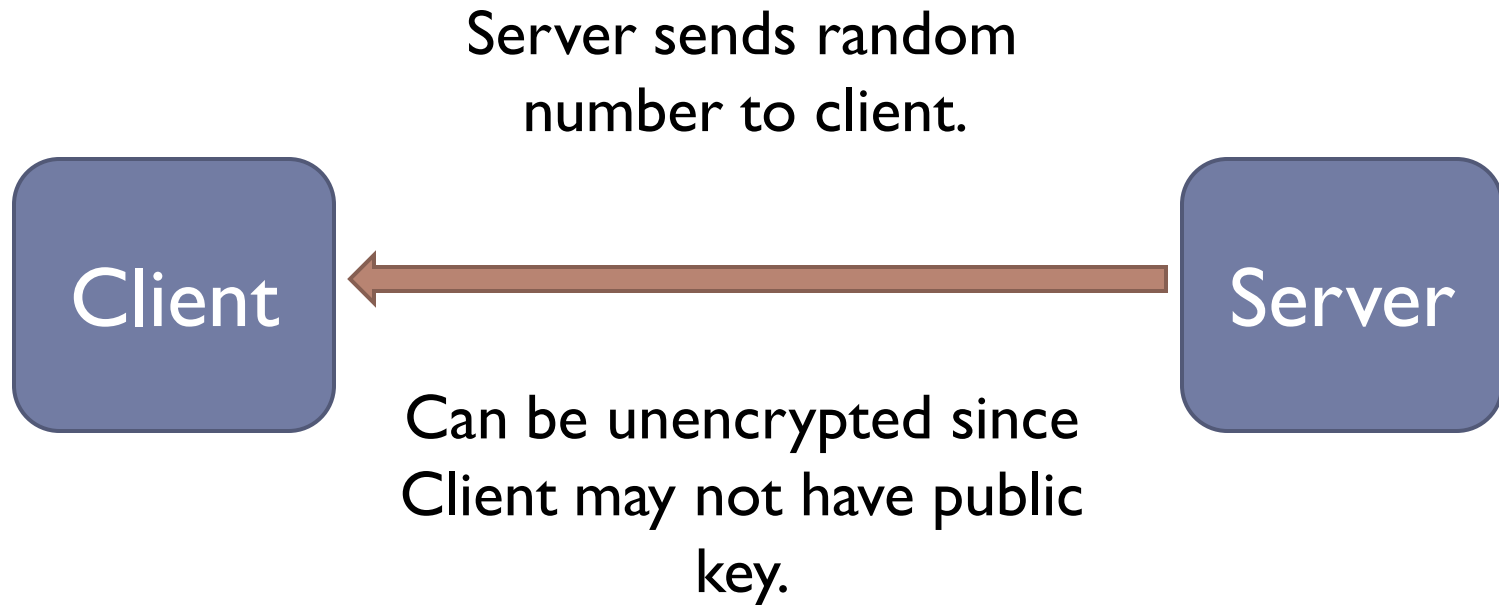
---

Sends random number  
encrypted with server's  
public key.



# How SSL works

---



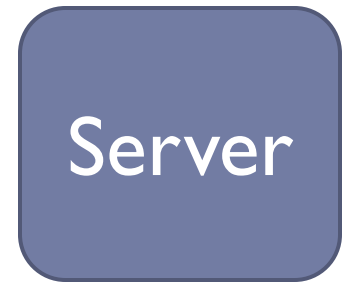
# How SSL works

---



94ABI34...

Server and Client compute a shared secret using the negotiated hash algorithm.



94ABI34...



# How SSL works

---

Communication is  
encrypted using the new  
shared secret & symmetric  
cryptography



# Client Authentication

---

- ▶ Server can authenticate the Client using its public key as well
- ▶ Requires key distribution
  - ▶ Server side must import every client public key into its keystore



# Limitations of SSL

---

- ▶ **Does not work well with intermediaries**
  - ▶ If you have a gateway handling SSL, how do you actually get the user information?
- ▶ **Limited ability for other authentication tokens beyond those of HTTP Auth**
  - ▶ i.e. SAML
  - ▶ Some implementations support NTLM (Commons HTTPClient)



# XML Signature & Encryption

---

- ▶ Provide message level security when needed
- ▶ Limited support across languages
  - ▶ Mostly Java & .NET
- ▶ Allows other types of authentication mechanisms beyond just SSL



# An XML digital signature

---

```
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm=
      "http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod Algorithm=
      "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference URI="#mySignedElement">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </ds:Transforms>
      <ds:DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmldsig#sha1" />
      <ds:DigestValue>EULddytSo1...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    BL8jdfToEb11/vXcMZNNjPOV...
  </ds:SignatureValue>
  <ds:KeyInfo>
    ...
  </ds:KeyInfo>
</ds:Signature>
```

---



# The Atom Publishing Protocol

# What is Atom?

---

- ▶ **Atom: a format for syndication**
  - ▶ Describes “lists of related information” – a.k.a. *feeds*
  - ▶ Feeds are composed of entries
- ▶ *User Extensible*
- ▶ More generic than just *blog stuff*



# The Bare Minimum Atom Feed

---

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

  <title>Example Feed</title>
  <link href="http://example.org/" />
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>RESTful Service Inc.</name>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b91c-0003939e0af6</id>

  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03" />
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <content>
      ... (content is optional)
    </content>
  </entry>

</feed>
```

---



# Atom retargeted for employee info

---

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

  <title>Employees</title>
  <link href="http://example.org/employee/info"/>
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>Acme Inc.</name>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>

  <entry>
    <title>John Doe</title>
    <link href="http://example.org/employee/info/dan_diephouse"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <content>
      <acme:EmployeeInfo>
        <acme:Name>Dan Diephouse</acme:Name>
        <acme:Department>Software Engineering</acme:Department>
      </acme:EmployeeInfo>
    </content>
  </entry>

</feed>
```

---



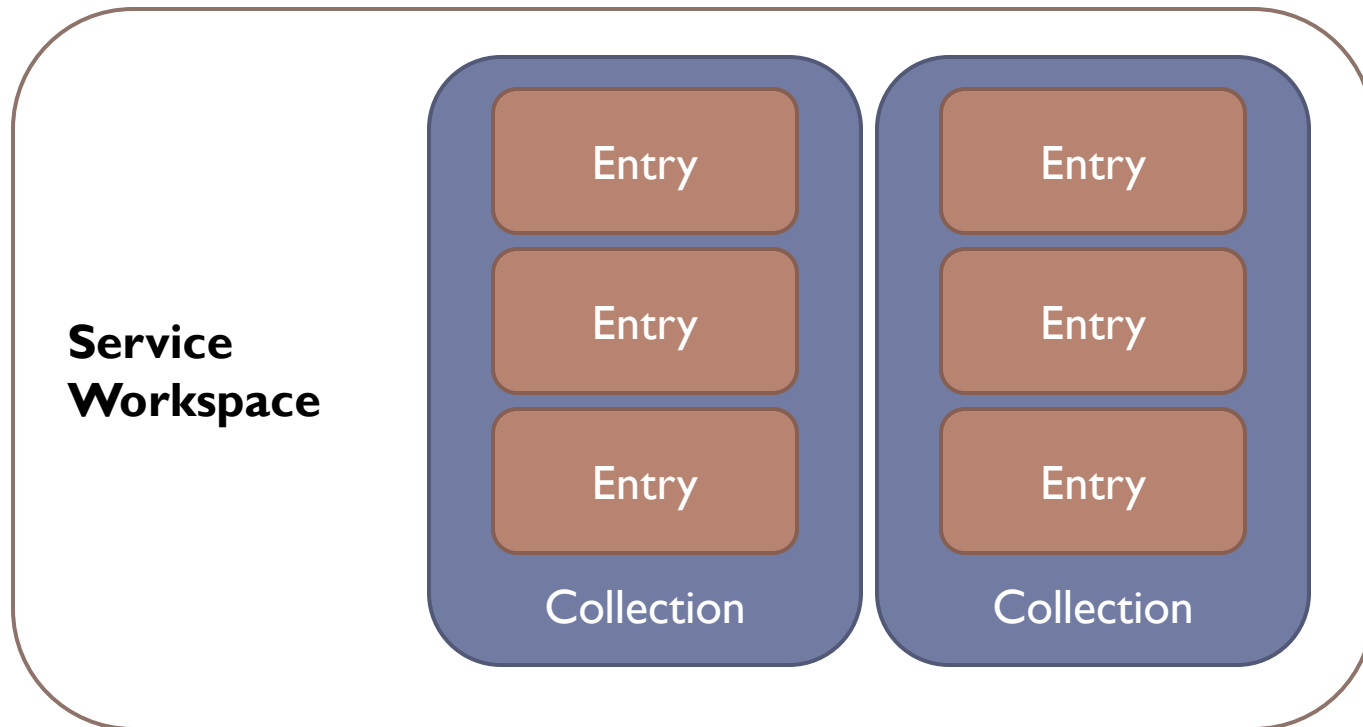
What does this have to do with REST?



# Atom Publishing Protocol

---

- ▶ RESTful protocol for building services
- ▶ Work with *collections* through a RESTful interface



# What is the Atom Publishing Protocol?

---

- ▶ Create, edit, delete feed entries
- ▶ Extensible Protocol
  - ▶ Paging extensions
  - ▶ Authentication extensions (i.e. Gdata)
  - ▶ Opensearch
- ▶ Properly uses HTTP so can be scalable, reliable and secure
- ▶ Implemented by a variety of clients and servers
  - ▶ Abdera, Amplee, blog stuff\*, etc



# Example: GET /

---

```
<service xmlns="http://www.w3.org/2007/app"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <workspace>
    <atom:title>Human Resources Site</atom:title>

    <collection
      href="http://example.org/employee/info">
      <atom:title>Employees</atom:title>
    </collection>

    <collection
      href="http://example.org/employee/pic">
      <atom:title>Employee Pictures</atom:title>
      <accept>image/png</accept>
      <accept>image/jpeg</accept>
      <accept>image/gif</accept>
    </collection>

  </workspace>
</service>
```

---



# Example: GET /employee/info

---

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Employees</title>
  <link href="http://example.org/employee/info"/>
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>Acme Inc.</name>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>

  <entry>
    <title>John Doe</title>
    <link href="http://example.org/employee/info/dan_diephouse"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <content>
      <acme:EmployeeInfo>
        <acme:Name>Dan Diephouse</acme:Name>
        <acme:Department>Software Engineering</acme:Department>
      </acme:EmployeeInfo>
    </content>
  </entry>
</feed>
```




## Example: GET /employee/info/dan\_diephouse

---

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>John Doe</title>
  <link
    href=".../employee/info/dan_diephouse" />
  <id>urn:uuid:...</id>
  <updated>2006-12-13T18:30:02Z</updated>
  <content>
    <acme:EmployeeInfo>
      <acme:Name>Dan Diephouse</acme:Name>
      <acme:Department>
        Software Engineering
      </acme:Department>
    </acme:EmployeeInfo>
  </content>
</entry>
```

---



# An employee leaving the company?

---

```
DELETE /employee/info/john_doe
```



# Why you should use APP for our app

---

- ▶ There are many APP implementations and they are known to work well together
- ▶ Atom format is well understood
- ▶ You can leverage existing solutions for security
  - ▶ HTTP Auth, WSSE, Google Login, XML Sig & Enc
- ▶ Eliminates the need for you to write a lot of server/client code
  - ▶ ETags, URLs, etc are all handled for you
- ▶ Integrates seamlessly with non-XML data



# What other tools are available for building RESTful applications?

---

## ▶ Java

- ▶ Servlets
- ▶ Restlets
- ▶ Spring MVC
- ▶ CXF
- ▶ Jersey (JSR 311 reference implementation)

## ▶ Ruby on Rails

## ▶ Python's Django

## ▶ Javascript's XMLHttpRequest 😊



# Limitations (Constraints) of REST & HTTP

# Limitations

---

- ▶ **HTTP is NOT an RPC or message passing system**
  - ▶ Not ideal for sending event based messages
  - ▶ May have performance constraints for asynchronous messaging that JMS/others may not have
- ▶ **Security Standards**
  - ▶ Most people will just use SSL, but...
  - ▶ Exchanging other types of authentication tokens is not possible unless they are custom HTTP headers
  - ▶ No standard way to establish trust relationships beside certificate hierarchies/webs



# Conclusions

---

- ▶ **HTTP Provides many tools/properties for us to build scalable, reliable, secure systems:**
  - ▶ Idempotent and safe methods
  - ▶ ETags/LastModified
  - ▶ Hypertext
  - ▶ Caching
  - ▶ URLs & Content Types
  - ▶ SSL
- ▶ **Beyond HTTP**
  - ▶ Atom Publishing Protocol
  - ▶ XML Signatures & Encryption
  - ▶ Much more... (Open ID, SAML, RDF, etc)



# Questions?

---

- ▶ **Blog:** <http://netzoid.com/blog>
- ▶ **Email:** [dan.diephouse@mulesource.com](mailto:dan.diephouse@mulesource.com)

