



The Memonic Architecture

Webtuesday

Chris Hauzenberger

Patrice Neff

chris@memonic.com

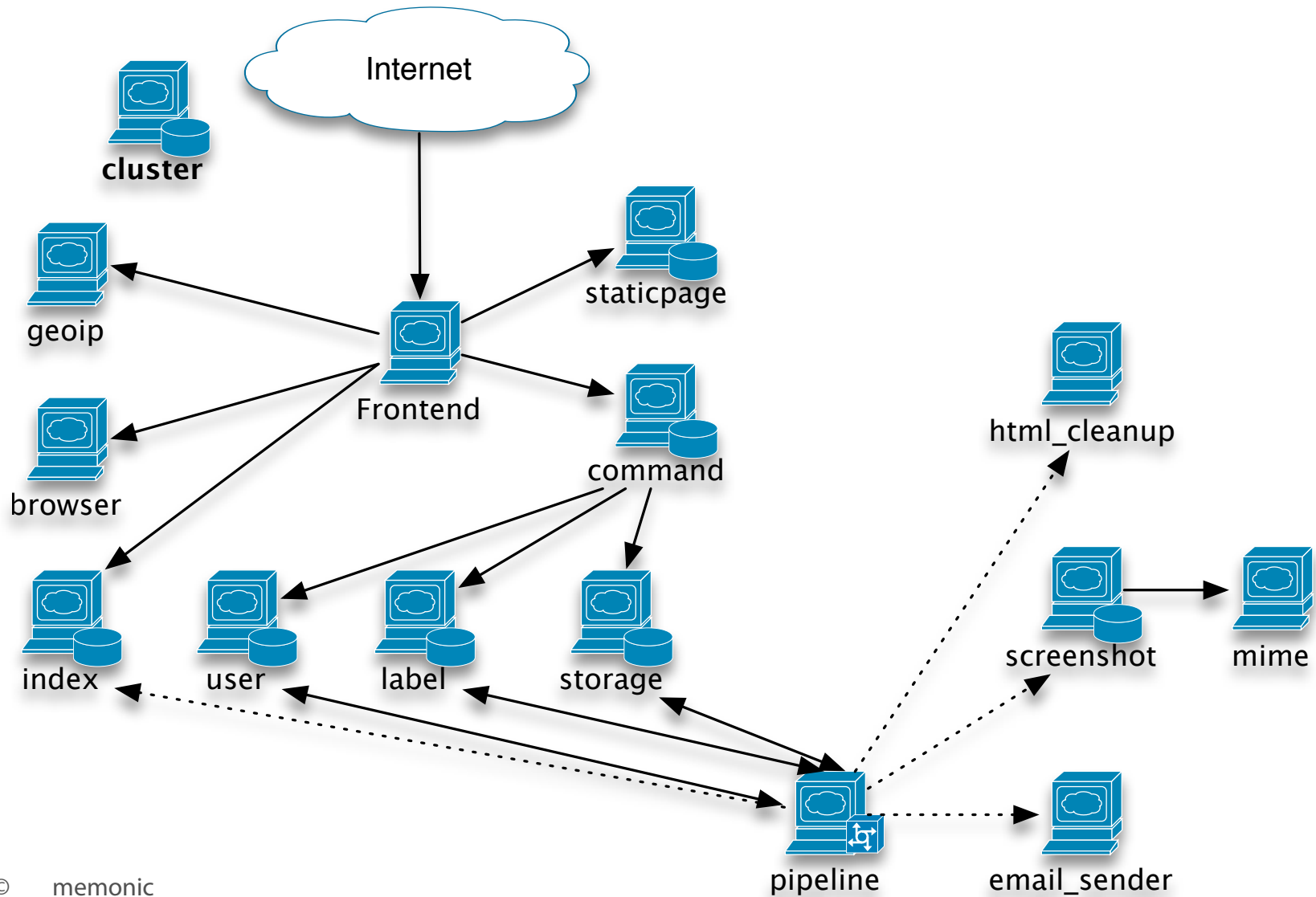
patrice@memonic.com

20100413

memonic



Overview





REST Services

- Dependencies local to service
 - Database
 - Queue
 - Synchronization
- Try the simplest service that works
- Services are the new classes?



Motivation for Services

- Modularity
 - Team separation
 - Clear boundaries
 - Easier migrations / replacements
 - Versioning
- Competence centre
- Re-use of services
- Best tool for the job
 - Programming language
 - Dependencies (Database, Queue, ...)
- Scalability



Showcase: Cluster

- User ID to cluster (shard)
- Assigns a user to cluster on first visit
- Challenge: anonymous users leave garbage

Bonus: Twitter's Gizzard seems to be a better version of our cluster service.



Showcase: Command

- Handles cluster lookups
- Stores undo tickets (X-Undo-Ticket response header)
- Undo operation with the ticket ID
- Clean up old undo ticket

Intent: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and **support undoable operations.**

(Gamma et. al. - Command pattern)



Showcase: Pipeline

- Asynchronous processing of dependencies
- Add additional data, create screenshot, index, send notifications, ...
- Uses AMQP internally - but HTTP externally

Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

(Gamma et. al. - Observer pattern)



Showcase: Simple services

- GeolP
- Browser
- MIME
- HTML cleanup
- Logo
- Screenshot
- ...



WsgiService example: GeoIP

```
import logging
import datetime
import pygeoip
from wsgiservice import *

log = logging.getLogger(__name__)

@mount('/1/{ip}')
@validate('ip', doc='The IP address to look up.',
         re='[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}')
class IpResource(Resource):
    """Represents the GeoIP information of a single IP address."""
    @expires(datetime.timedelta(days=365))
    def GET(self, ip):
        """Return the country code for the given IP address."""
        geoip = pygeoip.GeoIP()
        country_code = geoip.country_code_by_addr(ip)
        if not country_code:
            raise_404(self)
        log.debug("Mapped IP %s to country code %s", ip, country_code)
        return {'country_code': country_code}

app = get_app(globals())
```



Showcase: Data storage

- Storage
- Label
- Index
- User
- Subscription
- ...



Dealing with Services: RestClient

- Easy-to-use interface for finding and accessing services
- Resolves cluster
- Finds service
- Performs HTTP call



RestClient: Code

```
# GET request to command service
```

```
client.GET('command', ClosestCluster,  
  '/1/label_item/userxyz/inbox',  
  {'start': 1, 'count': 10},  
  headers={'some_key': 'some_value'})
```

```
# POST request to pipeline service
```

```
client.POST('pipeline', 'userabc',  
  '/1/relation/userabc/contacts', {'diff': diff})
```

```
# PUT request to label service
```

```
client.PUT('label', 'userhij',  
  '/1/label/userhij/somelabel',  
  {'title': 'some title'})
```



Locating Services

- User cluster
- Closest cluster
- Random cluster
- First cluster (first cluster that returns a successful response)
- All clusters

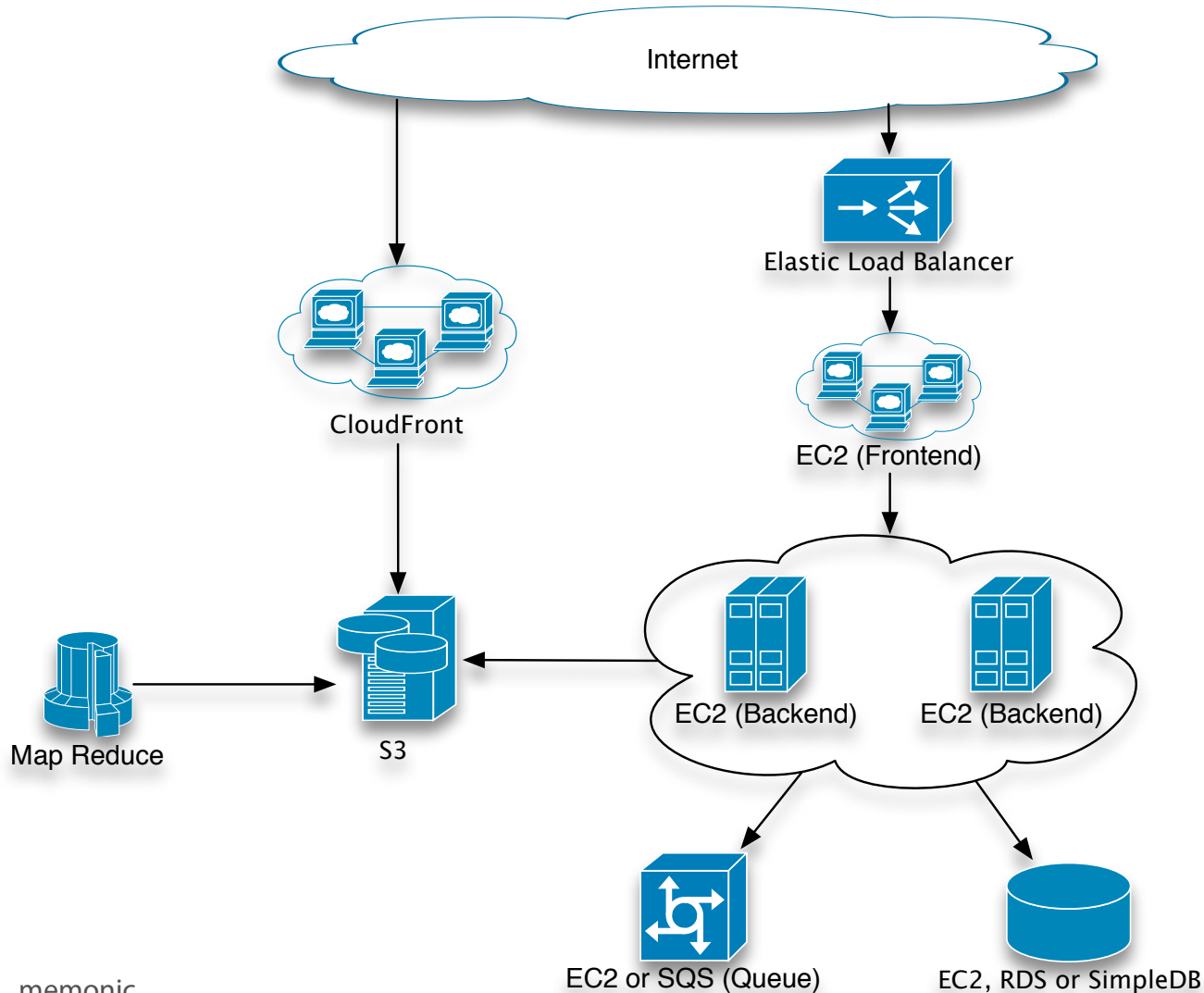


Statistics

- Each service exposes it's statistics
- Automatically added to Ganglia



Our usage of Amazon Web Services





Technology Components

- Dojo: JavaScript
- MySQL: Most data stores
- Pylons: Frontend applications
- Python
- RabbitMQ: Queue
- Solr: Fulltext search
- WsgiService: Services



System Setup and Deployment

- virtualenv
 - Isolated runtime environment for each service
- setuptools
 - Create installable packages (eggs)
 - Specify version and dependencies
- Puppet
 - Configuration management
 - Specify desired server state
 - Daemon assures that setup is complete



Puppet: Deploy Service

```
class service::geoip inherits service {  
    pythonwsgiservice::webpy { geoip: }  
    nginx::fcgi { geoip: port => 456,  
                 source_class => trusted; }  
}
```

```
class service::tinyurl inherits service {  
    pythonwsgiservice::webpy { tinyurl: }  
    nginx::fcgi { tinyurl: port => 123,  
                 source_class => trusted; }  
    mysql::database { "tinyurl": }  
    mysql::user { "tinyurl@localhost":  
                 password_hash => ...; }  
}
```



Puppet: Configure Build

Works for Python services

```
hudson::project {
  "frontend.shared.toolbar":
    directory => "frontend-shared/toolbar",
    package => "nektoon.frontend.shared.toolbar";
  "service.email_listener":
    directory => "service/email_listener",
    package => "nektoon.service.email_listener";
}
```

Works for Windows applications

```
hudson::windowsproject {
  "client.lib.apiabstraction":
    directory => "MemonicApiAbstraction",
    target => "Release",
    binaryname => "MemonicApiAbstraction.dll";
}
```



Tools

- Specification / Documentation: Confluence
- Issue Tracking: Jira
- Scrum Support: Greenhopper
- System Setup: Puppet
- Continuous Building: Hudson
- Monitoring: Nagios
- Statistics, Graphs: Ganglia
- Log Mining: Splunk



Links

- Jobs

<http://www.memonic.com/page/en/jobs>

- Scalability with HTTP

www.memonic.com/user/pneff/set/presentation-http-scalability

<http://mem.to/t/1Fsc>

- Memonic on AWS

www.memonic.com/user/pneff/set/presentation-cloud-swiss

<http://mem.to/t/1tMH>



Thank you!

Chris Hauzenberger

chris@memonic.com

<http://twitter.com/ch13>

Patrice Neff

patrice@memonic.com

<http://twitter.com/pneff>

20100413

memonic