Mission-Critical Cloud/Enterprise Hybrid Deployments

Eugene Ciurana
Open-source software evangelist
CIME Software Labs
http://eugeneciurana.com/contact
About Eugene…

- 15+ years of experience building mission-critical, high-availability systems infrastructure
- 12+ years of Java work
- Open-source evangelist
- Official adoption of open-source / Linux at Walmart Stores
- State-of-the-art tech for main-line of business roll-outs
  - Engaged by the largest companies in the world
  - Retail
  - Finance
  - Oil industry
- Background: industrial robotics to on-line retail systems
This Presentation Is About

- How to design, implement, and roll-out cloud/enterprise hybrid applications
- Different cloud architectures
  - PaaS
  - SaaS
  - Infrastructure
- Technologies: ESBs, clouds, mini-clouds, Java, App Engine, etc.
- How cloud services lower costs for the enterprise
- Understanding the advantages of
  - Deployment on the cloud
  - Hybrid deployment between the cloud and the enterprise
What You’ll Learn

● Identify the kinds of applications that best use cloud computing services

● Identify the advantages of using Platform-as-a-Service or Software-as-a-Service resources

● Learn the caveats of cross-language and cross-platform integration

● Learn high-performance alternatives to XML data serialization for data exchange

● Learn how to structure event-based, stateless apps for scalability
What is the Cloud Anyway?

- Ask 10 different people to get 10 different answers

- In general, you can use 4 types of cloud offerings
  - Platform as a Service
  - Software as a Service
  - Infrastructure as a Service
  - Pure infrastructure

- Some times you integrate prefabricated apps, some times platform, some times both
Characteristics of Cloud Services

- Quick deployment of prepackaged components
- Uses commodity, virtualized hardware and network resources
  - Amazon Elastic Cloud 2 (EC2) and Simple Storage Service (S3)
  - Google App Engine
  - Rackspace Cloud Services
- The overall model is “pay as you consume”
- Horizontal scalability is achieved by adding or removing resources as needed
- Can host full applications or only services
Characteristics of Cloud Services

- They can replace the data center
- Basic administration moves to the application owner
  - It may move away from the IT team - political fallout
- For the bean counters... it’s an operational expense!
  - Tax advantages
  - Turn on and off as needed
  - In a tight economy, IT infrastructure ends up under the CFO - give the guy options
- Assuming sensible SLAs and system availability, the ROI is better than for co-location facilities or company-owned data centres
About Scalability and High Availability

● What do ‘scalability’ and ‘high availability’ mean?

● Scalability: the property of a system to handle bigger amounts of work or to be easily expanded in response to increased demand
  • Horizontal
  • Vertical

● Availability: how the system provides useful resources over a set period of time
  • Uptime != availability
  • Many factors affect availability, such as network, storage, processor, etc.
SLAs, Availability, and the Cloud

- Service Level Agreements (SLAs) drive the architectural and technological decisions

- Cloud systems’ scalability characteristics are often pitched

- What is the availability?
  - What is the impact to the business?
  - How do you perform disaster recovery?

- What service level are the vendors offering?
  - The higher the availability, the higher the cost
  - Co-lo, data centre deployments still make more sense for three-nines availability and above
SLAs, Availability, and the Cloud

<table>
<thead>
<tr>
<th>Availability</th>
<th>Downtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>36.5 days</td>
</tr>
<tr>
<td>99%</td>
<td>4 days</td>
</tr>
<tr>
<td>99.9%</td>
<td>9 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>53 minutes</td>
</tr>
<tr>
<td>99.999%</td>
<td>5.3 minutes</td>
</tr>
<tr>
<td>99.9999%</td>
<td>32 seconds</td>
</tr>
</tbody>
</table>

- **Amazon S3 Outage**
  - Major companies affected
    - eBay
    - LeapFrog Enterprises software downloads
    - Apple iTunes Store
  - Amazon’s SLA? “Sorry, it won’t happen again.”
Platform and Software as a Service

Once you establish availability requirements, identify the type of implementation

- From a vendor?
- In-house development?

Software as a Service (SaaS)

- Vendor provides a full application and is responsible for scaling it and meeting SLAs
- Company integrates with vendor via web services or traditional EDI
- Examples: GSI Commerce, Salesforce.com

Platform as a Service (PaaS)

- The vendor provides a virtual operating system and/or application stack
- The vendor provides a well-defined API set for interacting with the system
- Examples: Amazon Web Services, Google App Engine, Nirvanix
PaaS Systems

- **Amazon EC2 and S3**
  - Virtual images with Linux or Windows, app server, database, etc.
  - Billed by the hour + bandwidth
  - Java, PHP, Ruby, other technologies supported
  - “Data centre in the sky”
  - Lower cost than Rackspace or Nirvanix, weaker SLAs

- **Google App Engine**
  - The apps are written as callbacks from a virtualized service
    - Datastore ~= database; Memcache ~= caching; all interaction over HTTP
  - Billed on usage
  - Python only; RPC over HTTP/HTTPS only
  - “You run on the same infrastructure as Google”
  - Low cost, still evolving, minimal SLAs
Typical Scalable Cloud Architecture

Amazon S3
http://media.company.com

Google App Engine
http://www.company.com

User

Firewall

services.company.com

Enterprise Service Bus

Node
Node
Node
Node

Legacy Back-end

Enterprise Database
Typical Scalable Cloud Architecture

A few things to notice...

- All the mission-critical systems are still behind the firewall
- The cloud is used for high load applications and services
- Cloud apps work independently of the data centre apps, and vice versa.

- Loose coupling over web services
- Assume that the data in the cloud is “throw away”, either consolidated or sourced in the data centre
- Cloud is mostly used for distribution and scalability
Typical Scalable Cloud Architecture

End-User System (Mac, Windows)

LeapFrog Connect

Web Browser

S3 Content Repository

Third-party Partner Site

www.leapfrog.com

connected products

LearningPath

Internet

Firewall

Mule ESB backbone

HTTP, SOAP (CXF), REST, etc. routing, filtering, and dispatching; ActiveMQ JMS broker; dedicated LeapFrog services

Mule ESB tailbone

Connected products SOAP, REST web services

Customer Data

Game play Data

Mule ESB funnybone

Device log upload, processing, servlet container

Servlets App Logic

Device Logs

Content Management System

REST, JCR

Content Authoring

Crowd SSO

User Credentials
Which Parts Go To The Cloud?

- **Internet**
- **Load Balancer**
  - **Application Server**
    - Tomcat 6
  - **Services Proxy**
  - **Application Server**
    - Tomcat 6
  - **Load Balancer - Backbone**
    - Mule ESB 1.6.2
    - Mule ESB 1.6.2
    - Mule ESB 1.6.2
    - Mule ESB 1.6.2
  - **Load Balancer - Tailbone**
    - Mule ESB SOAP, REST
    - Database
  - **Load Balancer - Funnybone**
    - Mule ESB SOAP, REST
    - Mule ESB servlet, MTOM
    - NFS share
  - **Load Balancer - Message Broker**
    - Mule ESB servlet, MTOM
    - ActiveMQ
    - NFS share

Backbone - message filtering, routing, dispatching, queuing, events
Which Parts Go To The Cloud?

- Depending on cloud configuration, load balancing may not be necessary
  - Amazon provides “elastic IP addresses”
  - Google App Engine provides stateless calls only
    - Application state is kept by the client and/or Memcache
  - Rackspace provides either “elastic” IP or explicit load balancers

- Enterprise Service Bus and other integration may be split between the cloud and the data centre
  - Message routing OK if latency is below SLA minimum requirements
  - Strategic partner, application, services integration may occur 100% on the cloud, data consolidation behind the firewall
Case Study

- Large consumer software and services company
- .Net / Windows servers infrastructure
  - ASP.Net
  - SQL Server
- Two tier architecture
  - The system “just grew” and became a business
- Serves pages, services, and media
- 6-month scalability project begun February 2009
  - Implementation
Case Study - Objectives

- Stable architecture available by July 1, 2009
- Low cost
- Build scalability wherever possible
- Optimal data transfer rate for all properties
  - Web systems
  - Media assets
- Meet business requirements and SLAs
  - Hard dates set independently of the technology team
  - Milestones: July 1 and October 10, 2009
Case Study - Initial Configuration

- Grew “as needed” without planning
- Combines end-user and internal applications in the same server
- Only scales vertically
- Applications and databases are tightly coupled
- Application servers and services are tightly coupled
- Single environment: from dev to prod without passing Go and collecting your $200!
- Disaster recovery? 4 hours minimum from backups
Case Study - Initial Configuration

- Client
- Client
- Client
- Zone Server
- Internet
- IRC
- Admin .Net
- Applications .Net
- SQL Server
Case Study - Phase I Scalability

- Offers limited scalability by separating application services from web applications
  - Middleware and front-end scale independently
- Enforces separation of concerns by requiring app developers to use a services layer for implementation
  - .Net, PHP development from the front-end OKi
  - Third-party development integration OKi
  - Enforces well-documented APIs
  - Enforces normalized data exchange protocols
  - Expedites development because changes may be unit tested independently in each layer
Case Study - Phase I Scalability

- Introduces a CDN for asset delivery
  - Amazon S3 (optional Cloud Front) for asset delivery
  - Reduces load on company servers and bandwidth costs
  - Implements S3rsync for customer updates

- Introduces database replication for production environments

- Establishes a continuous integration environment
  - Set tools for a hybrid UNIX/Windows environment
  - Better build/release process
Case Study - Phase I Scalability
Case Study - Phase I Scalability

- Failover with traditional database replication techniques

Diagram:
- Primary Cluster
  - Node 0
  - Node 1
- ESB as app services provider
  - Partition 0
    - DB 0
      - DB 0b
  - Partition 1
    - DB 1
      - DB 1b
Case Study - Phase II Cloud Deployment

- Web-based applications move to PHP and/or Wicket
- The database and stored procedures migrate to Oracle or PostgreSQL
- The Mule ESB layer does away with many (most?) stored procedures and implements business logic as stateless POJOs/UMOs
- Software stack is “best of breed”, a combination of commercial and open-source software
Case Study - Phase II Cloud Deployment

- Web and other RPC services must coexist
  - Different partners use different protocols
  - Mule transformers take care of all the interfacing so that development may scale / continue independently of what happens in other layers

- Bandwidth is the expensive portion of this equation!

- Data exchange protocols
  - Clients: free-for-all: custom, .Net serialized, XML, JSON
  - In-cloud server images: protocol buffers
  - Cloud-to-data-centre: protocol buffers
  - Data centre: protocol buffers, XML

- Replication strategy: data centre - cloud untrustworthy
Case Study - Phase II Cloud Deployment

- Internet
- Zone Server
- Client
- Client
- Client
- Client
- CDN
- AWS - EC2
- All Web Properties
  - server
  - server
  - server
  - server
- services HTTP, other proxy
- Load balancer
- Mule ESB
- Mule ESB
- CMS
- Terracotta or Coherence
- Databases
- replication
- Databases
- Load balancer
- Crowd SSO
- Crowd SSO
Case Study - Phase II Cloud Deployment

Production

Cloud Cluster
- L1 Cache
- App Server
- Databases
- Applications
- Load balancer
- Mule ESB
- Terracotta or Coherence
- Crowd SSD
- Crowd SSO

QA

Eucalyptus Cloud Cluster
- L1 Cache
- App Server
- Databases
- Applications
- Load balancer
- Mule ESB
- Terracotta or Coherence
- Crowd SSO

Development

Eucalyptus Cloud Cluster
- Databases
- Mule ESB
- Terracotta or Coherence
- Crowd SSO
- Developer 0
- Developer 1
- Developer 2
- Developer 3

Staging

Cloud Cluster
- L1 Cache
- App Server
- Databases
- Applications
- Load balancer
- Mule ESB
- Terracotta or Coherence
- Crowd SSD
- Crowd SSO

Version Control System, Continuous Integration Server, Automatic Builds, Unit Tests

CDN

Game Server

Web Applications

Services HTTP other proxy
Case Study - Phase II Cloud Deployment

### Traditional Hosted Environments

<table>
<thead>
<tr>
<th>Environment</th>
<th>Setup</th>
<th>Monthly</th>
<th>Bandwidth</th>
<th>Total / year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>$1,300</td>
<td>$5,580</td>
<td>$2,000</td>
<td>$70,260</td>
</tr>
<tr>
<td>Staging</td>
<td>$1,300</td>
<td>$5,580</td>
<td>$0</td>
<td>$68,260</td>
</tr>
<tr>
<td>QA</td>
<td>$600</td>
<td>$1,450</td>
<td>$0</td>
<td>$18,000</td>
</tr>
<tr>
<td>Development</td>
<td>$600</td>
<td>$1,450</td>
<td>$0</td>
<td>$18,000</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$174,520</strong></td>
</tr>
</tbody>
</table>

### Elastic Computing Cloud

<table>
<thead>
<tr>
<th>Environment</th>
<th>Setup</th>
<th>Monthly</th>
<th>Bandwidth</th>
<th>Total/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>$4,225</td>
<td>$284</td>
<td>$5,950.00</td>
<td>$21,805.00</td>
</tr>
<tr>
<td>Staging</td>
<td>$4,225</td>
<td>$284</td>
<td>$1,750.00</td>
<td>$12,633.00</td>
</tr>
<tr>
<td>QA</td>
<td>$2,275</td>
<td>$154</td>
<td>$1,750.00</td>
<td>$15,261.00</td>
</tr>
<tr>
<td>CI</td>
<td>$2,600</td>
<td>$720</td>
<td>$2,500</td>
<td>$5,820.00</td>
</tr>
<tr>
<td>Development</td>
<td>$2,275</td>
<td>$154</td>
<td>$1,750.00</td>
<td>$6,462.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$61,981.00</strong></td>
</tr>
</tbody>
</table>
Wanna know more about real life cloud, scalable systems?

Subscribe to the newsletter!

http://eugeneciururana.com/scalablesystems

Questions?

Thanks for coming!

Eugene Ciurana
Open-source software evangelist
CIME Software Labs
http://eugeneciururana.com