Designing Cloud Solutions

Objectives
Key objectives of this chapter ■ Cloud design strategies ■ The concept of layering ■ Analysis and design best practices

1.1 Getting Started …

1.2 Implications of Vendor Lock-In

- The lack of standards in the Cloud platform and Cloud infrastructure space results in a huge dependence upon vendor implementations
  - Open standards currently focus upon messaging, presentation, and systems integration
  - Open standards for Cloud infrastructure are in development, but we are still very early in this space

- This results in huge implications for your initial vendor selection decisions
  - You cannot count on portability across Cloud vendors at this point
  - The greatest danger, exists with SaaS and PaaS solutions (IaaS solutions are all relatively similar, with the exception of their data storage mechanisms)

1.3 Dealing with Vendor-specific Service API

- You need to decouple service interface used in your cloud-based solutions from vendor-specific implementation
- Front vendor cloud's service API with a facade interface to hide implementation details
  - Use *Dependency Injection* for decoupling interface with the actual service implementation (use standard JEE6 Context Dependency Injection or other inversion of control containers, e.g. Spring)

1.4 Know Your Cloud Application's Needs

- Applications have different needs for resources:
  - CPU, memory, storage, I/O and networking
  - Use in-house or cloud deployment benchmarks to identify your application's resource orientation
- Knowing the technical aspects of your cloud applications help with the right technical architecture and the selection of run-time systems with matching parameters
  - For example, CPU-bound applications would require more powerful types of virtual machines (with more vCPUs and RAM)

1.5 Data Physics

- Data Physics considers the relationship of data and the processing / computing elements that use the data
- There is a cost to moving data
  - Data should be located as close to the processing point as possible (that's what Hadoop does when starting its MapReduce jobs)
    - Ideally, data should be processed on the same machine where it sits (you have no control over this, though -- most cloud storage solutions use some sort of network attached type - NAS, SAN, iSCSI, etc.)
- In modern networked environments, it may be cheaper to move data to another node then to persist it
- Architects need to be able to maintain well-defined relationship between processing units and the data they process

1.6 Cloud Design Strategies

- Designs of cloud solutions should aim to take full advantage of the target cloud platform's capabilities, such as
  - Elastic resource provisioning
  - A full spectrum of *-as-a-Service functionality
  - The possibility of establishing Hybrid cloud links between on-premise corporate data centers and public clouds to support cloud bursting and other use cases

1.7 Designing for Failure

- In the Cloud, everything can fail: server, storage, network, you name it ...

- Design for system resiliency to failures
- Create failure containment boundaries that would block error propagation
- Prepare for cloud infrastructure downtime and upgrades
- Define the exit strategy

1.8 Designing for Cloud Availability

- High availability (HA) characteristics of an application are achieved by creating a cluster of machines in the same application tier and fronting the cluster with a load balancer(s)
- AWS offer Availability Zones (AZ) which are isolated from each other and are connected through low-latency fiber-optic network links to support fast data transfer between them. When you create your virtual instances, you can specify an AZ for them to run. So, you just need to create your EC2 instances in different AZ's and front them with an Elastic Load Balancer

  Note: There are two or more AZ's within a single AWS geographical region

1.9 Designing for Cloud Scalability

- (Horizontal) Scalability is achieved by enlisting additional resources to cope with the increased service demand
- Normally, the topology of cloud solutions takes advantage of the auto-scaling service and load balancing component(s) offered by the hosting cloud
- The auto-scaling capability allows cloud customers to scale their computing and other capacities up or down automatically according to pre-defined resource utilization thresholds
- With auto scaling, the number of resources (e.g. virtual server instances)

provisioned at run-time transparently increases and decreases to meet demand peaks and valleys

- Among other benefits, auto scaling helps with minimizing cloud costs
- AWS Auto Scaling Service uses information provided by Amazon CloudWatch monitoring service and is free to use (clients only pay Amazon CloudWatch fees.)

Notes:

Normally, the auto-scaling service is a client of a monitoring and resource utilization tracking system that collects and serves target metrics to its clients.

1.10 UI Considerations

- Design for device-independence as cloud-based solutions are generally consumed by multiple agents: desktop browsers, mobile phones, tablets, etc.
- To fully realize the potential of the cloud-based solutions, the web UI should be designed to provide an optimal viewing experience by your clients. Such techniques include:
  - Responsive Web Design (RWD) techniques that are used to allow easy reading and navigation with a minimum of page resizing and scrolling across a wide range of devices (from desktop computer monitors to mobile phones)
  - "Progressive enhancement" which predates RWD and uses the strategy of starting with page design targeting mobile devices (that don't understand CSS and JavaScript) and incrementally (progressively) add "unobtrusive JavaScript and CSS" progressive enhancements

1.11 Analyzing Cloud Requirements

- Analysis is done after the definition of the problem statement or problem domain
- Functional requirements will need to be translated into system interfaces
- Non-functional requirements will need to be supported by service contracts and captured in quality of service attributes

1.12 "Good/Not-so-Good" Use Cases for the Cloud

- "Good" use cases for the Cloud:
  - Transient (run & throw away) applications (testing, PoC apps)
  - Variable workload (public web sites)
  - Computationally extensive, distributed data computing tasks (Hadoop)
- "Not-so-good" use cases for the Cloud:
  - Vertically integrated enterprise applications
  - ERP systems

1.13 Design the Cloud Service Interface

- Interoperability is the key
  - Basic Profile 1.0 compliant web services
  - RESTful services
- Open Standards
  - OCCI (Open Cloud Computing Interface)
  - OCC (Open Cloud Consortium)

- The Open Group

- Use the community's feedback to help design service interfaces

Notes:

Open Cloud Computing Interface (OCCI) model is built on top of Resource Oriented Architecture (ROA) and uses REST web services to handle client requests for such services as Virtual Machine deployment, cloud management requests, monitoring queries, etc.

1.14 Designing for Cloud Non-Functional Requirements

- Those are in support of existing functional requirements
- They do not perform a business function, but still required by the problem statement
- Must be documented and enforced

- Common Non-Functional Requirements include:
  - security
  - scalability
  - availability/reliability
  - performance
  - configurability
  - extensibility

1.15 Practical Observations and Rules

- Data is king
- Data outlives applications

- Applications outlive integrations

1.16 Selecting the Right Object Storage

- **S3**
  - Real-time data retrieval
  - Objects sizes up to 5 TB
  - Programming interfaces:
    - CLI / REST / SOAP
  - In-place encryption (optional)
  - Possibility to expose some content as static Web content (images)
  - Content sharing across cloud infrastructure
- **Glacier** Archival Object Storage Glacier in AWS
  - Low-cost alternative to S3 for data that is infrequently accessed
  - Slow access times (1 - 5 hours)

1.17 Analysis and Design (A&D) Best Practices

- There are a number of best practices that are applicable to cloud applications and services (and, in part, to conventional enterprise systems as well) that can be loosely grouped in the following categories:
  - Prototyping
  - System Partitioning
  - Treating data as resources (accessible through RESTful endpoints)
  - Leveraging cloud platform services
  - Using asynchronous communication patterns

- Designing for failure
- Caching
- Staying hands-on

1.18 A&D Best Practices - Prototyping

- If possible, create a quick prototype (in the cloud)
  - This, among other things, may help fine-tune existing requirements, introduce new ones or drop those which are not realistic
- Sometimes, it is not a prototype, but a *pretotype,* that would help avoid a product or application failure
  - Pretotypes help make sure you're building the right "it" before you build it right

1.19 A&D Best Practices – System Partitioning

- Partition the system into more granular modules and components that can be potentially deployed on different parts (tiers) of your cloud solution
  - That would improve system security, maintainability and other non-functional system requirements
- Front various complex system domains with a (web-based) facade interface to ease system integration, hide complexity and promote SOA principles
- Identify and isolate areas of possible system instability by wrapping them up with a stable interface (with predictable response and availability metrics)
- Assess the applicability of the Anti-corruption layer concept from the Domain-Driven Design (*http://en.wikipedia.org/wiki/Domain-driven_design*)

to your solution

1.20 A&D Best Practices - Leveraging Cloud Platform Services

- Clarify vendor relationships and leverage your cloud platform's capabilities exposed as application, storage, and technical services
- Is your application interruption-tolerant (meaning it is not mission critical and can tolerate service interruptions)? If so, see if the vendor offers more cost-effective server rates for deploying such applications
  - EC2 Spot Instances can be a good fit for deployment of such applications

Notes:

EC2 Spot Instances can significantly lower your computing costs for time-flexible, interruption-tolerant tasks.

Spot Instances allow you to name your own price for Amazon EC2 computing capacity. You simply bid on spare Amazon EC2 instances and run them whenever your bid exceeds the current Spot Price, which varies in real-time based on supply and demand. The Spot Instance pricing model is providing potentially the most cost-effective option for obtaining compute capacity, depending on your application.

(http://aws.amazon.com/ec2/spot-instances)

1.21 A&D Best Practices - Using Asynchronous Communication Patterns

- Using asynchronous communication patterns will help you meet performance and scalability requirements (more on the next slide: *MOM to the Rescue*)
- When used in messaging, asynchronous communication pattern does not require the message sender and receiver be available at the same time

- Explore the value proposition of the Staged Event-Driven Architecture (SEDA) systems

Notes:

The staged event-driven architecture (SEDA) refers to an approach to software architecture that decomposes a complex, event-driven application into a set of stages connected by queues. It avoids the high overhead associated with thread-based concurrency models, and decouples event and thread scheduling from application logic. By performing admission control on each event queue, the service can be well-conditioned to load, preventing resources from being overcommitted when demand exceeds service capacity.

(http://en.wikipedia.org/wiki/Staged_event-driven_architecture)

1.22 MOM to the Rescue

- One way to ensure scalability and proper service performance levels is to use cloud message-oriented middleware (MOM) where requests with different priorities can be placed in different queues and processed on a priority basis
- Cloud vendors offer these ready-to-use MOM solutions:
  - AWS offers Amazon Simple Queue Service (SQS) that provides an efficient and scalable MOM platform
- If needed, deploy external MOM systems (e.g. RabbitMQ)

1.23 A&D Best Practices - Preempt Possible Data Corruption

- Design you data storage solutions in a way that have built-in data corruption and inconsistency prevention mechanisms
- eBay preempts possible data corruption and prevents partially constructed objects in their Master – Detail database using the following data insertion sequence:

- Detail records are inserted first, followed by the Master database inserts
- If there were a failure in the middle of the process, the database will just have orphaned records in the detail table (which can be purged later)
- Should it be done in reverse, the system may end up being in an inconsistent Master (updated first) – Details (never updated due to a failure in the middle of transaction) relationship

1.24 A&D Best Practices - Caching

- In-memory cache allows you to
  - Improve your application's performance
  - Avoid unnecessary hits on back-end systems and generation of the same content more than once
- Use cache where possible
  - Decide on the cache eviction algorithm (least recently used, least used, etc.)
  - Configure cache expiration rules (every unit of time, on-demand, on-event)
  - Some caches offer a persistence option – explore it
- Production-ready caching solutions you can leverage: *Memcached, Redis*
  - *Redis* can help with access to complex data structures (collections, etc.)
- Leverage Amazon ElasticCache
  - Fully managed in-memory data store with sub-millisecond latency
  - Compatible with Redis and Memcached

1.25 A&D Best Practices - Staying Hands-On

- See what technology solutions and techniques other people use - this may help with your design work:
  - Efficient data transfer techniques (which also help with cross-language interoperability): Avro, Protocol Buffers, Thrift
  - NoSQL systems
  - Messaging systems: RabbitMQ, ZeroMQ
  - Specific server systems: Node.js
  - etc.

Notes:

Google Protocol Buffer (*protobuf*) data encoding system may help you in many ways:

- It will generate data bindings for a number of programming languages
- Data structure changes are transparent for clients; new attributes are safely ignored
- It offers high-performance data un/marshaling (~ 200 MB/s)
- You can cost-efficiently persist protobuf-encoded data transfer objects
  - When saving protobuf-encoded data transfer objects, you can also avoid data transformation step if the clients fetching the data at a later time are also protobuf format aware

1.26 Be Aware of the CAP Theorem Constraints

- When using NoSQL systems, be aware of the CAP theorem constraints formulated by Eric Brewer
  - http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- It states that any distributed computer system can have at most two of three desirable properties:

- **C** - consistency equivalent to having a single up-to-date copy of the data
- **A** - high availability of that data (for updates)
- **P** - tolerance to network partitions

- In cloud distributed environments, **P** is a given, leaving designers with a choice between **A** (availability) and **C** (consistency)
  - In the end, it is the business decision (which normally results in an eventually consistent persistence store)

1.27 The CAP Triangle

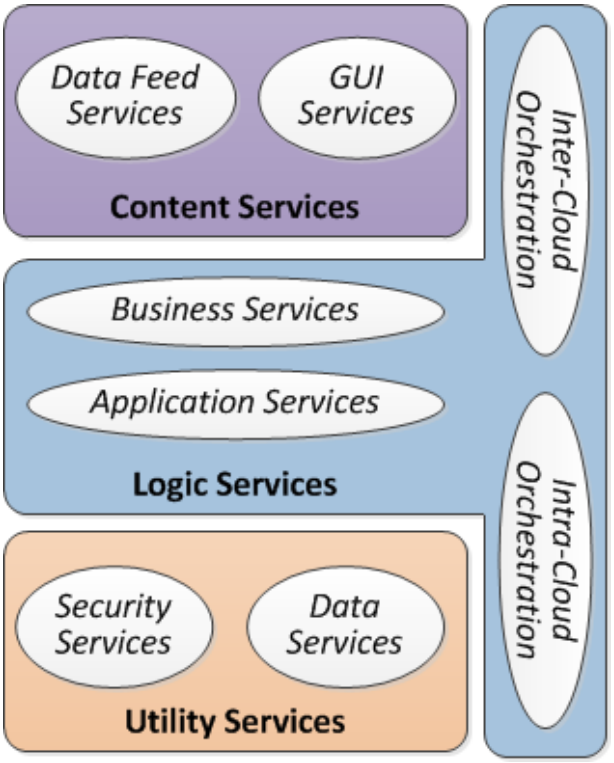
Source: Eugene Ciurana, QCon2013

1.28 Cloud Layering

- The application stack of cloud solutions is often made up of various layers
- Cloud layers help partition you application into logically cohesive parts promoting modular design principles
  - *Note*: Cloud layers are modeled after the layer concept used in conventional enterprise application designs
- Layering aids in creating loosely-coupled, scalable and cost-effective cloud-based solutions

1.29 Cloud Layering Overview

- Content services: Abstracts access from clients to data. This typically includes services called directly by a consumer that may result in many different services coming together to perform the action
- Logic services: Abstracts data from applications and houses functionality and logic components. Also supports orchestration both within (intra) and between (inter) clouds
- Utility services: Abstracts applications from servers and servers from storage



Notes:

Cloud layers are logical, not necessarily physical

It is important to understand that these cloud layers are logical constructs. Not every provider will necessarily use this exact terminology and many of the solutions out there offer a hybrid of these different capabilities.

Cloud solutions are often composites

Many times a final cloud solution is a composite of various layers. You might rely upon *utility services* from an IaaS vendor, build *application services*, and then consume some *content services* in the form of data feeds from a 3rd party content provider. All three of these types of services might combine to produce a single solution stack that is exposed to your customers as a content service that you provide to them.

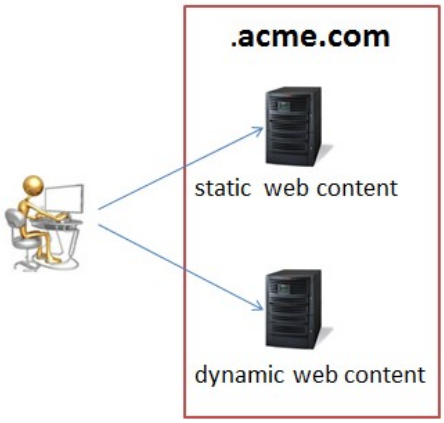
1.30 Content Services

- Content Services are often exposed directly to end-users. These might be consumed by mobile apps, web browsers, or server-side applications that incorporate these services into mashup-style application content
- Two primary types of content services
  - **Data Feed services** – Typically exposed by RSS / ATOM for syndication or perhaps via adhoc APIs (JSON, SOAP, Pure HTTP, etc.)
    - ✔ *Examples: blog feeds, news feeds, Google Maps API, Amazon S3 API*
  - **GUI services** – Consists of HTML-style snippets as well as rich media widgets via Flash or streaming media
    - ✔ *Examples: picture services, audio / video streams, Amazon eCommerce API (product catalog subset)*

1.31 Separate Static Content from Dynamic Content

- To help with scalability and systems maintenance, partition your web

application along the static and dynamic content boundary



- If possible, push out dynamic content generation down to the client's browser (RIA using AngularJS, React, etc.)

1.32 Logic Services

- This is where logic is housed for handling requests made by users. There are several categories of service that exist in this portion of a cloud solution
  - **Business services** – coarse-grained, often composite services that provide industry-centric and domain-centric capabilities
    - *Examples: Amazon eCommerce API, PayPal API, Member management services, Claim management services, Product management services*
  - **Application services** – fine-grained services that provide access to discreet functionality
    - *Examples: Search services, Weather services, Postal services, Calculation services*
  - **Orchestration services** – coordinate multiple services together into a flexible process flow *(explored in greater detail next...)*

- ✔ *Examples: Procurement process, enrollment process, claim management process, provisioning process*

Notes:

Composite Services

Many times services will be combined to perform more complex tasks. This especially happens with business services. Composite services combine application and business services together to execute common combinations of requests

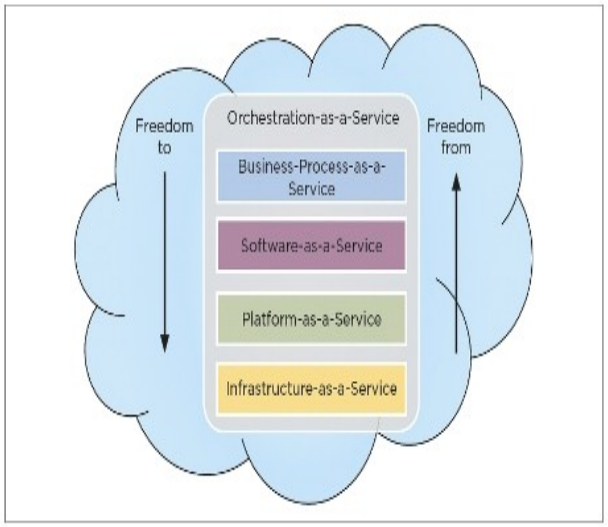
Composite Services can consist of:

- locally developed and deployed services
- cloud based services
- locally developed, but remotely deployed services

Orchestration vs Composite Services

This is discussed next, stay tuned...

1.33 Orchestration in the Cloud



- There will be not just one Cloud but a number of different sorts: private Clouds and public ones, which themselves will divide into general-purpose and specialized ones
- The term “InterCloud” means a federation of all kinds of Clouds, in the same way that the internet is a network of networks. And all of those Clouds will be full of applications and services
- One way of weaving these pieces together is through orchestration
- What is needed?
  - An Assembly and Enterprise Cloud Orchestration layer in the Cloud to fully deliver useful business advantages

Notes:

Orchestration vs Composite Services

A more elaborate form of composition comes in the form of orchestration. Composition is handled programmatically, whereas orchestration involves a declarative model of weaving pieces together. To change the behavior of a composite requires refactoring the code and is more prone to error. Changing the behavior of an orchestration merely requires modifying the declarative logic and is typically less prone to error since the underlying pieces remain intact.

1.34 Designing for Cloud Security - OWASP 10

- Provide security audit of you Web tier based on OWASP 10 top security projects (https://www.owasp.org/index.php/Top_10_2013-Top_10) which get reviewed and updated every year:
- A1-Injection
  - Hostile code/data injection, such as SQL, OS, and LDAP, sent as a harmful command
- A2-Broken Authentication and Session Management
  - These application defects may allow attackers to compromise user passwords, keys and session tokens and assume their identities
- A3-Cross-Site Scripting (XSS)
  - Usage of untrusted and unvalidated data sent to a web browser can allow attackers to execute harmful scripts in the victim's browser
- A4-Insecure Direct Object References
  - Unprotected access to a system resource that allows unauthorized access
- A5-Security Misconfiguration
  - A wide range of security configuration flaws

1.35 Designing for Cloud Security - OWASP 10 (Cont'd)

- A6-Sensitive Data Exposure
  - Inadequate protection for sensitive data (health records, SIN numbers, etc.). Such information needs extra level of protection such as encryption at rest or in transit
- A7-Missing Function Level Access Control

- ACL is not properly configured or enforced

- A8-Cross-Site Request Forgery (CSRF)
  - The victim's browser is tricked into sending a forged HTTP request as programmed by the attacker
- A9-Using Components with Known Vulnerabilities
  - Known vulnerabilities of libraries, frameworks, and other software components can be exploited to run harmful system commands with full privileges which can lead to server takeover
- A10-Unvalidated Redirects and Forwards
  - Without proper URL validation, attackers can redirect victims' browsers to phishing or malware sites, etc.

1.36 Designing for Cloud Security – Multi-Factor Security

- For boosting authentication process strength, some cloud vendors offer Multi-Factor Authentication (MFA)
- MFA provides an extra level of security that you can apply to your cloud environment
- You will have a physical device (similar to the RSA SecurID authenticator/fob) or a virtual MFA app installed in your mobile phone that has your additional credentials
- With MFA enabled, users signing on to their cloud environments, must provide their regular username and password (the *first factor* – what they know), as well as an authentication code from their MFA device (the *second factor* – what they have). Taken together, these multiple factors provide increased security for your cloud account settings and resources

1.37 Stepping Across Site Silos

- Currently, there are a number of user identity sharing techniques that help clients cross sites silos
  - SAML
  - OpenID
  - OAuth

1.38 Stepping Across Site Silos – the SAML Protocol

- SAML is an SSO protocol
  - Predominantly used in the Enterprise space
  - Complex (and expensive) to implement
  - Toolkits are few (e.g. SAML toolkit for Ruby on Rails)

1.39 Stepping Across Site Silos – the OpenID Protocol

- Like SAML, the OpenID protocol falls under the SSO classification
  - Auto-discovery of identity provider feature
    - The OpenID (URI) of Scott.Fitzgerald@gatsby.com states that Scott's identity provider to contact is gatsby.com
    - Using OpenID URI is more secure than submitting actual credentials
  - Simple
    - There exists a wide range of toolkits for setting up OpenID in your web site
  - Wide Internet and cloud adoption (Google, et al)
  - Due to lack of enterprise class OpenID providers is considered

"consumer" grade

- Disadvantage: applications need to know every user's OpenID

1.40 SAML vs OpenID

	OpenID	SAML
Service provider initiated SSO	x	x
Identity provider initiated SSO		x
Identity provider discovery	configured per user	configured per account
Just-in-time-provisioning	via SReg	directly
Performance	slower	faster
Positioning	consumer	enterprise

Source: https://onelogin.zendesk.com/entries/270738-OpenID-or-SAML-for-enterprise-SSO-

1.41 Stepping Across Site Silos – OAuth

- OAuth is a token-based authentication and authorization protocol that allows sharing of resources between web sites without exposing users' passwords but rather submitting a secret token shared between the participating sites
  - The token has a time-to-live interval and authorization rights granted by the user
- OAuth 2.0 is considered by many not suitable for enterprise-level integration
  - OAuth 2.0 completely relies on SSL for confidentiality and server authentication and doesn't support signature, encryption, or client identity verification
  - Protocol implementations vary in quality

- However, Google supports OAuth 2.0 as the recommended authentication mechanism for all of its APIs

1.42 Utility - Security Services

- Security applications delivered as cloud-based services will have a dramatic impact on the industry, as many cloud-based services will more than triple in many security segments, according to Gartner, Inc.
- Benefits:
  - *the ability to obtain more enterprise security controls or functions on demand*
  - *more vendors to offer their security products as a service and quickly match the IT service delivery infrastructure — such as bandwidth, storage and processing — to the demand for their as-a-service delivery*
- Warning:
  - *Enterprises will need to prioritize the adoption of encryption technologies that provide easy movement to longer keys*

Notes:

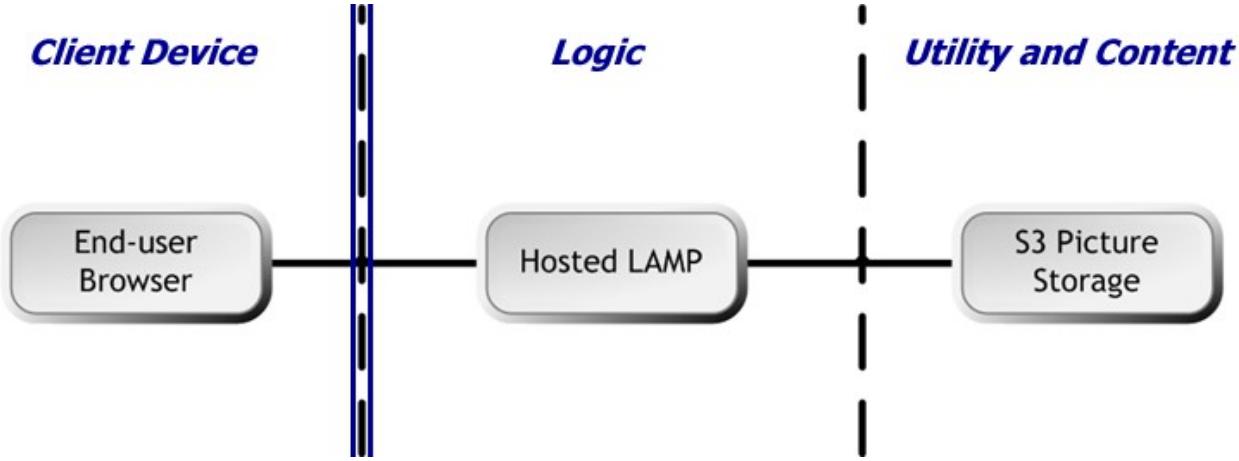
Source: http://www.gartner.com/it/page.jsp?id=722307

1.43 Out-of-the-Box Security Service Example

- AWS's CloudHSM (Cloud Hardware Security Module) Service (http://aws.amazon.com/cloudhsm) is a representative example of cloud platform-wide security service
- CloudHSM uses a tamper-resistant hardware appliance deployed within the AWS cloud (their data centers) that provides secure key storage and cryptographic operations

- CloudHSM allows you to securely generate, store and manage your cryptographic keys used for encrypting your data
- CloudHSM supports a variety of use cases and applications, such as database encryption, Digital Rights Management (DRM), Public Key Infrastructure (PKI), authentication and authorization, document signing, etc.
- It has been validated to comply with government standards for secure key management
- Access to cryptographic keys is open only to the owner

1.44 Simple Layering Example



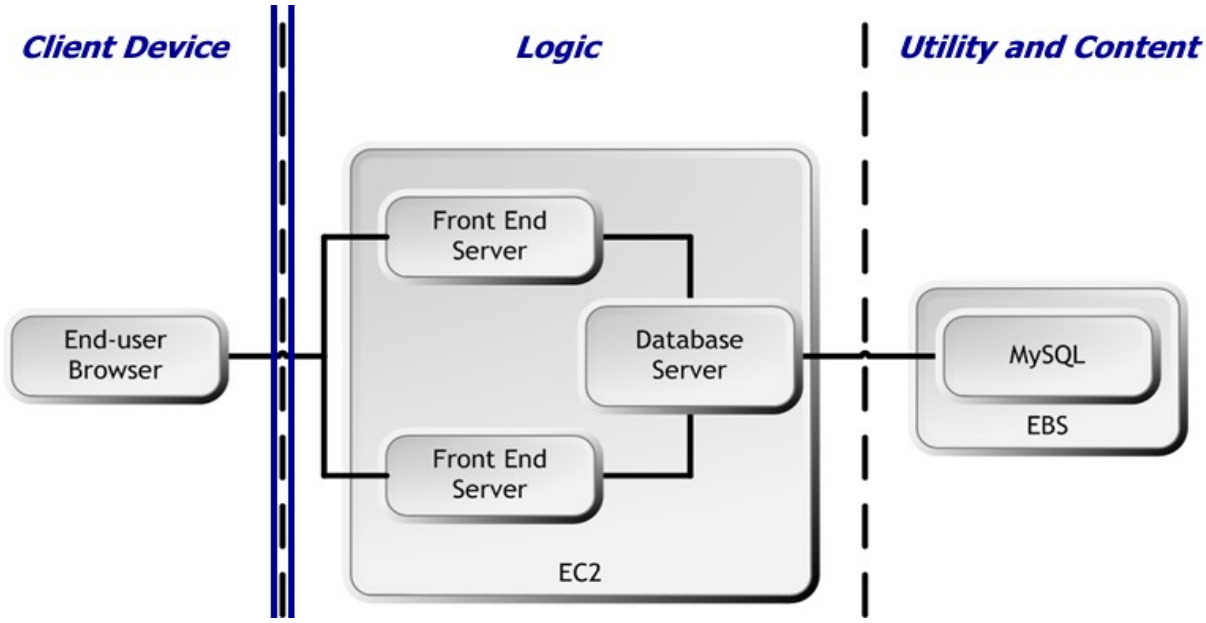
Hosted Server with IaaS Storage *(Utility Service)*

Notes:

In this simple example we have hosted an open source application (LAMP = Linux, Apache, MySql, PHP) with a third party and then connected to Amazon's S3 to provide us with a Utility Service.

SOURCE: This example and the subsequent examples are adapted from the book: *Cloud Computing Explained* are used, with permission, by John Rhoton.

1.45 Layering Example with Dedicated IaaS

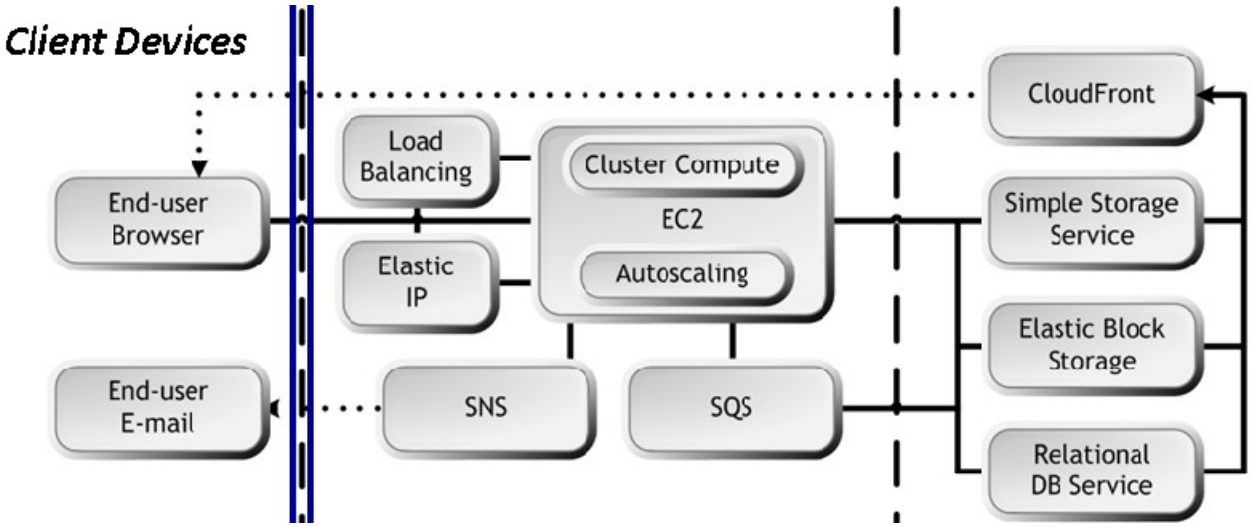


Dedicated IaaS *(Logic and Utility Services)*

Notes:

Here we have a pure Amazon AWS solution utilizing EC2 and EBS.

1.46 Amazon Web Services Integration Diagram



1.47 Handling Error Messages in the Cloud

- There are two types of error codes: client and server (including infrastructure)
  - Client error codes suggest that the error was caused by something the client did, such as an authentication failure or an invalid AMI identifier
    - In the SOAP API, These error codes are represented as faults and are prefixed with Client. For example: Client.AuthFailure
    - In the Query API, these errors are accompanied by a 400-series HTTP response code
  - Server error codes suggest a server-side issue caused the error and should be reported
    - In the SOAP API, these error codes are represented as faults and are prefixed with Server. For example: Server.Unavailable
    - In the Query API, these errors are accompanied by a 500-series

HTTP response code

Notes:

Summary of AWS Client Error Codes:
http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/api-error-codes.html

1.48 Designing for Cloud Maintainability

- Cloud systems, after being deployed, continue to evolve over time
- Maintenance involves fixing bugs (about 20% of the effort) and improving performance or other system attributes and generally enhancing the system (80% of the effort)
- So, the cloud solution design should aim to help with the 80% of the maintenance work
  - For that, design should take into consideration:
    - ✔ system architecture (partitioning, services, etc.)
    - ✔ system maintenance helpers (centralized configuration management, logging facility, integration with monitoring systems)
    - ✔ impact analysis of each solution part's failure
    - ✔ deletion of eventually obsolete capabilities and their replacement

1.49 Summary

- When designing for the cloud
  - Be aware of data physics implications
  - Leverage SAML, OpenID or OAuth authentication protocols to deal with authentication challenges in the cloud
  - Multi-Factor Authentication (MFA) hardens the authentication process in the cloud by providing an extra level of security
  - Concept of Layering
  - Content, Logic, and Utility Services
  - Application, Business and Composite Services
  - Orchestration in the Cloud