ECM3408 Enterprise Computing Lecture Notes

Ogaday Willers Moore
Department of Computer Science
School of Engineering and Computer Science
University of Exeter

2015/04/09

Contents

1   Introduction

1.1   What is Enterprise Computing?

Enterprise computing is all about combining separate applications, services and processes into a unified system that is greater than the sum of its parts.

> "Enterprise Computing is seen as an integrated solution/platform for separate business problems, such as database management, analytics, reports, which have previously been treated as single business problems treated with specific software solutions."

- `http://www.techopedia.com/definition/27854/enterprise-computing`

1.2   Drivers and Motivations

Up-scaling Enterprises need to be able to upscale to suit an increase in customers.

Outsourcing Specialist service providers will be able to fulfill requirements cheaper and easier than in-house solutions. Circumvents the need to invest in hardware, software and human resources, which requires *Capital Investment*.

1.3   Assumptions

This course makes two axiomatic assumotions.

1. **Enterprise Computing is based on web technologies.** This is not a controversial claim because of the many examples are out there. Netflix, Google, Twitter, Facebook are all worth billions of dollars and are based upon web technologies. Businesses will all need to provide their services and reach their customers over the web in one capacity or another.

2. **Enterprise Computing is done on the Ruby on Rails framework.** This is a controversial claim because there are many models available for web enterprises and Ruby on Rails provides just one instance of that. However this course will only be concerned with Enterprise Computing with Ruby on Rails.

2   Three Tier Architecture

Web programs often follow a *Three Tier Architecture*.

Data Services Tier This provides database access. Having this tier provides abstraction from the database behind the business. This means that migrating services, expanding onto more suitable hardware and scaling up is easier.

Business Services Tier This provides the "business logic", which is the functionality of the enterprise. For instance, Amazon provides for items to be added to carts, carts to be checked out, products to be searched and displayed and many many more small services which comprise their webshop and are all functions of the business. This might require communicating with the data services tier and for the output to be sent to the presentation tier.

Presentation Services Tier This provides the user interface and means that the responses can be adjusted for each user. Specific users can be targeted differently with specialised content; either suitable presentation for a device and browser can be provided (see headers). For example, mobile websites for mobile devices and html generated for internet explorer instead of firefox can be sent to the client. Alternatively, specific demographics can be targeted with specific presentational services. Facebook, Google and Amazon all often provide experimental new interfaces to a subset of their users. This is made possible by the abstraction of the Presentation tier, which can be altered without affecting or interfering with the underlying business and data tiers.

2.1 HTTP: The Hypertext Transfer Protocol

Communication is provided through the Hypertext Transfer Protocol (HTTP). HTTP is a call and response protocol. The client makes a call (Request) and the server sends a response. The response is usually a web page or service. All HTTP communication is in form text (Strings), rather than ints, bools etc.

2.1.1 HTTP Requests

HTTP Requests consist of a request line, request headers and a request body.

The Request Line The request line consists of a method (or verb), a resource and a version. eg: METHOD RESOURCE VERSION GET /images/logo.png HTTP/1.1

The request methods are GET - for receiving data POST - for sending data/send amendment data PUT - for amending data/send replacement data DELETE - for deleting data N.B. GET requests print the data in the URL, which is really bad for anything except debugging. Therefore bank details which are sent with a GET request, for instance, will leave those details in the browser in plain text, so either the request can be intercepted and read, or the browser can later be scraped because there will be a record of the url, and the url, along with the details, will be recorded in server log files.

Request Headers Headers consist of Name: Value pairs on separate lines. eg. Content-Length : 255

The Request Body The request body is empty except for the case of POST requests, which send the data in the body.

N.B. The server can also deny your request! It is under obligation to follow the request, so for instance DELETE google.com HTTP/1.1 would not work.

2.1.2 HTTP Responses

HTTP responses consist of status line, response headers and a response body.

Status Line The status line consists of version, code and reason eg. HTTP/1.1 200 OK The string "reason" is determined by the code. The code indicates different responses from the server, and the first digit classifies the response. 2 - good 3 - relocation advice, resource has moved. 4 - bad ie. 404 resource not found. 5 - transient bad, server overloaded etc.

Response Header Response headers consist of Name : Value pairs again. Cache-control : caching allowed Connection : Connection preferred Content Length : (length of body)

A case where caching might not be preferred is a live score ticker, so it will continue to update and wont use the cached value of the score. N.B. Can get a snapshot of past browser activity by looking at the browser cache.

Response Body This is often in HTML, but can be anything else (javascript, xml, etc).

3   Introduction to Ruby

Ruby is duck typed, interpreted, high level, OOP capable language. It was invented/designed by Yukihiro "Matz" Matsumoto in Japan around two decades ago and has risen to popularity because of its expressivity and Ruby on Rails, a very successful web framework for building web apps. Ruby is introduced by countless high quality tutorials which I will not try to match.

4   Model-View-Controller

4.1   Design Patterns

Design patterns come about when people find a solution to a challenge and encode them in a pattern. What makes you stand out is the ability to not only learn from your own mistakes, but from others' mistakes as well by using their design patterns.

In the three tier architecture, a common design pattern is the Model-View-Controller which is a pattern upon which the Business tier can be built. In this course, the Model-View-Controller design pattern principles are the "rails" upon which we are sat.

4.2   The Model

The model is responsible for database access. Why use a model to access the db? Because then the pattern is independent of database architecture, or at least insulated from the databse and then the enterprise is more portable to different architectures and DB services.

4.3   The View

Responsible for data presentation. By separating the presentation from the data handling, it is easier to focus on creating the best presentation. It is possible to have visualisation experts to specifically work on the view. The view is very important because gives customers strong impressions of the enterprise, and can influence customer decision. For

instance, Amazon and Netflix provide different views in some cases in order gauge the effect on customers. Facebook and Google can upgrade and modify their interfaces without risking the other aspects of their service.

4.4   The Controller

Responsible for data processing and business logic. This section sends emails, adds things to cart, does programmatic things.

4.5   The MVC Pattern

This pattern is also reproduced in other languages and frameworks, such as PHP, and most e-commerce platforms follow this design pattern.

See the lecture slides for an example of an implementation of the MVC design pattern in Ruby on Rails.

5   Service Oriented Architecture

Pull away from traditional web development course to think about what makes enterprise web systems as opposed to simple web applications.

The whole thrust of enterprise computing is: What happens when things get big? When you have a large enterprise, you cannot have a single one of anything anymore.

In the three tier architecture, the Presentation and the Business tiers wont do much hard work - All the heavy lifting is done by the Data services tier. Thats why a load balancer is introduced, to make sure the work is evenly distributed across the multiple databases.

Business need multiple databases as they scale for two reasons:

Performance As customer numbers increase, the business must have increased capacity to serve their customers.

Redundancy Business need to protect against fault or attacks which bring down their services. If a server fails they will be unable to serve their customers, which is a cardianl sin in the world of business. As a business scales and gets more custom, redudancy needs to be provided.

5.1   Websites and Web services

Websites act as you expect: human users can navigate their browsers to them and interact in the browser. Web services are strongly correlated with APIs, as they can be queried directly, rather than being rendered with html like a website.

Decompose the services in order to provide understandability, isolation, uniform access(Everyone knows how it works, should all work the same way between services. Should be predictable. This provides a contract between user and service provider - cannot change API functionality once it is released or user will be upset and confused), scalability, and redundancy and failover.

Examples: Both models use the Service Oriented architecture with services decomposed. The advantages are written above, plus with the modularity of the structure of the business, it would be easy to expand to offer a new service as the business develops, such as offering a new insurance field, or a new service, plus if one service fails, the others will be unaffected. Another good example is Google; Google offers many diverse and well developed services (Ads, search, maps, mail, drive, google+, calendar, scholar, videos, the play store and many more) which are well separated and modulated. This is not always possible though, sometimes services are impossible to extract and insulate from the other services. A good example is the login service offered by a lot of enterprises. If that fails, a lot of the other services may be rendered unavailable.

6 Ruby on Rails

Ruby by itself is nothing special, does everything that python, php etc does. New generation of programming languages. We are going to look at a very useful framework for ruby though, Ruby on Rails. Does a lot of the low level stuff for you. Two advantages:

Recap MVC design pattern. Rails provides an outline implementation of the MVC in Ruby. Doesnt create the whole platform for you, you cant get a replicate facebook, for instance. However, far fewer mistakes, plus more maintainable, because the common frameworks means new developers can approach the project and see how it works because things are done in a standard way.

See lecture slides for tutorial. Can start with any component of the MVC.

"$rake" is the japanese / ruby equivalent of unix "$make" which is something to do with build automation. "$rake db:rake" means updates the latest version of the db. Uses "Object relational mapping" - There is a direct map between lines in the database and objects? ORM is another design pattern that allows databases to be represented in the code and manipulated in the code even if the types seem incompatible.

Access to urls will run ruby code. - known as "routing"

(submitting form is a bad idea - despite the example)

update and delete/destroy are not described in the view because they cannot be issued by the browser.

Might not need forms because the code might be accessed via other code/machines using urls like an API.

7 Remote Procedure Calls

Problem of how to distribute work between multiple CPUs. Necessary because the limits of serial processors are quickly reached these days.

A regular procedure call is simply a local method invocation with different args. ie. f(a, b, c).

Problems: Message passing: blocking/non blocking messages, race conditions. Naming problem: name not only args, but the machine upon which the code runs. Serialisation problem: Have to be able to send the data across a network. Does it make sense to do it? The cost of message passing is much higher than local memory access/computation.

Traditional RPC calls client needs a stub, which is what procedures the server provide. Both require a network layer for packing and unpacking args.

Problems with traditional model: Typing issues/formatting problem might not be checked until the message is at the server. Other problem: the client must have an implementation of the stub, remote procedure calls etc in order to be able to run it. Also slow.

Solutions: Use a standard interface. There are two particular ones: CORBA and RMI (for Java).

Were not going to use either. A much more widely available one is HTTP, consisting of the request line headers, bodies etc. as seen earlier. Even the above two are not ubiquitous and require the client, who cant be considered universally capable enough to download and install/update software such as CORBA.

Operation Information

We can specify the procedure to be called on the server in the resource section of the request. Can specify arguments as query string values when using GET. Follows ? character, as in google, youtube etc. Problem: can leave out arguments etc because HTTP is weakly typed. No strong checking of the imminent procedure call, might only be found at run time, and fail on the server. Advantage: gets through firewalls and only dependency is a modern browser, which are HTTP capable.

HTTP RPC Returns

Needs to be a significant amount of work: an RPC to add two numbers would not be worth it. 404 errors occur because of the weak typing of HTTP requests/RPCs. The APIs are not published in a machine readable way. Recall 400s are permanent and 500s are transient.

Close-Coupling

If you invent your own API you get a close coupling of server and client. Therefore, if the API changes, the contract is broken and the client programs will no longer work because their HTTP requests will fail.

Client sides will have to update their programs to reflect the changed server protocols. This is expensive, and can cause downtime, so customers and traffic might be lost. As stated before, changing APIs is best avoided.

8   Representational State Transfer

Recall: RPC. Arguments are marshalled (packed up) and unmarshalled (unpacked). Problem is there isnt a universal WSDL - way of describing which services are provided in an xml file which can

We would like our web services to be used by other programs as well as by humans. Services should be automatically accessible.

Where does REST come from? Ron Fielding tried to study and extract from the web why it was so successful. See lecture slides for list of things which made it successful.

Client server constraint. Server deals with services and data, and client provides UI, which means programs didn't need to be downloaded to provide the service. (Apps revisit this model though, which are successful because there is a direct link/contract with the app provider and downloader). Users browse services provided on the server.

Statelessness. Each time a client accesses a server it is as if for the first time. Use cookies to provide state client side instead. Advantage is that if the server becomes overwhelmed, an alternative server, in a load balanced system for instance, can provide the same service to the customer. Without cookies, this element would be fatal to the internet.

Caching. Headers dictate whether data can be cached or not. Means some repeated requests are not necessary. The cache obviates long routes to servers for data which has already been requested.

Uniform interface. Similar to databases, resources are managed using only CRUD. Interesting because delete cannot be uttered by any web browser.

Layered System. The internal layers such as databases cannot be seen by the client. One url is used to access a service, regardless of the architecture beneath it. Gets rid of close coupling: the client doesn't need to know the details of the architecture beneath, so the same service can be provided in the same fashion, even if the implementation is different.

Code-on-Demand. Extensibility. The server must be able to provide code to clients. In practice extremely difficult to pull this off. Current system is javascript. Big difficulty is arbitrary code execution is dangerous, so the code on demand must run in a sandbox - very hard to get right.

Web is about manipulating data on a server. Normally only view data, like GET.

If we want to produce a REST version of airport code look ups.

Advantage to using design patterns such as using the MVC architecture is that other people have made the mistakes and refined to process ahead of you, even if you would eventually discover the design pattern yourself.

Controller POST corresponds to create. "permit" is pertaining to only accepting argument which correspond to :code and :name GET corresponds to show. PUT corresponds to update. DELETE corresponds to destroy.

This means there is no need to provide an elaborate API. Routings are a translation file in ruby which tells the server which ruby code to run when it receives an HTTP request.

View Consists of markup/client side code such as html.

Doesn't need all the html elements which you might otherwise want. Don't submit information using code.

Mixing of code and html is a design pattern: the Template design pattern.

Finally: the world wide web has some incredibly successful ideas behind it.

9   Resource Oriented Architecture

Why do we want scalability in enterprise computing? Because we want our enterprise to be able to grow to match demand and the internet so far does not appear to have a near limit on how far it can scale, so our enterprise needs to be able to (potentially) match that. We dont want to reach a point of growth and find the architecture needs to be redesigned because that is costly and stops the service from reaching the customer.

REST constraints lead to: separation of resource from representation: the representation is given to the client, and it can be customised according to the client via headers, if it wants to. A spanish client might want the webpage in spanish, for instance.

The classic example is a coke machine, which is solid. Some professors decided to put the state of vending machine on the internet, and the representation was a view of that state.

1962 playboy centrefold is a image processing benchmark. Another example is google maps. Can see the change of resource as the view scales. The change between different resolution/scale images can be quite drastic.

self-descriptive representations: Heres the message, and heres how you interpret it! - not always the latter half, sometimes its just I hope you have an application that can open this! ie. pdfs. Often done it XML which will be covered later. This is the metadata describing the syntax and semantics.

Machines can read XML because it is a formal language it can easily process with tags containing different aspects which are easy to navigate. Included is a handy description of how to make sense of it.

Why dont we ship pictures instead of xml? Its the same information. Because the picture is much more difficult to work with than the xml/raw code/information. So much structure to wade through in pdfs/images. So representations are much easier to deal with by the client side machine. It is better to send fundamental representations than convoluted ones.

manipulation of resources by representation: Representations can be changed. Google maps encouraged people to add pins to it to mark out locations and shops. People like direct manipulation. The above would not have worked if the pins had to be entered via latitude and longitude.

Also much easier to change: rather than having to manipulate machine code to change a variable, can change the representation and it will change the resource! hypermedia as the engine of application state: The big idea! HATEOAS. There are two ways of telling a server what to do next: Either click a hypertext url (link) or submit a form with the submit button. The advantage: The client keeps track of state! Imagine if google had to keep track of the state of each of their users, the memory cost alone would be massive, never mind the book keeping. Also, the client can reconnect to another server as determined by a load balancer, but that doesnt matter! There is no need to go back to the same one. The same server could crash, and another server on the other side of the world could take over and it wouldnt make a difference, because the state is stored by the client, in the browser history and url etc.

In the example, if someone asks for fanta, the server doesnt remember if you asked for fanta before! It doesnt care if you looked at the menu or not.

Even if the server could manage the memory constraints and book keeping of storing state, its not particularly failure/crash safe at all! if the server goes down, so does the state.

10   Storage as a service

To recap, if we build with the contents of the above lectures in mind, then we can be ensured to dodge a few pitfalls ahead in the road of a expanding enterprise.

Microsofts SkyDrive, cloud driven storage service was taken to court by Sky television, so its a relic nowadays.

Coffee bar is an analogy for the MVC. Barrista takes your order and has various hardware to accomplish the order. They may need to access their storage for some ingredients. Just before it is served they ask for special needs such as sprinkles.

Why use MVC? Because its been proven to work, but also because its is compulsory with Ruby on Rails.

Use cases correspond to http verbs, ie. GET, POST, PUT, DELETE, used for RESTful web services.

Object Relational Map hides db statements behind objects. So you call ruby code which automatically calls the sql commands.

Template design pattern allows the mixing of html and ruby code, only differentiable by the tags.

Have the feedback such as Created! so the user knows if something has gone wrong.

Curl interacts with http requests? look into.

new is for human beings, not for robots.

11   Extensible Mark Up Language

What is mark up?

12   JSON

Why even consider JSON? XML is Verbose. JSON, comparably is sparse. Even bytes saved can make a big difference if you have many customers/clients. The limit is on the number of bytes you want to send, not on the customers. Want to be able to deal with potentially infinite customers. Originally invented/invented to be used for the asynchronous web.

Synchronous web: Two ways of stirring the server: URLs/links/hypertext and form submit buttons. Problem with synchronous web: latency or delay. Lots of idle time during which computation time could be done.

Asynchronous web client makes requests when events occur, ie. telling you if a username is taken. This is all done in javascript. Or when google suggests search terms or lists results before you have finished typing.

Javascript runs in the browser, server program runs too. The requests and responses can be sent in any format, but we want to send it in an easily parsed format. Eg the JSON format.

AJAX: Asynchronous Javascript and XML machinery.

Why is JSON easier to parse than XML? Because of the tags: many more characters for the same functionality/meaning. eg. open array is [vs ¡catalogue¿, etc. Therefore JSON is quicker to read and write.

AJAX controls both ends of the server/client communication, so you will not receive any unexpected requests.

JSON schemas describe the structure of a document with declarative rules. A message which receives a DTD is self describing. Not used in AJAX, because unnecessary. Schemas are an afterthought that attempt to replicate one of the successful features of XML, the DTD.

In industries there can be decided upon formats. Very powerful because it is a common language, which means business to business transactions are easier.

13   Database Integrity

Integrity of data is knowing that the database contains what you think it contains, ie. correct and uncorrupted data.

What is the greatest strength of an enterprise? Not its people, its the data., and one must ensure its accuracy or integrity. Dunhumby is the company which does loyalty/memberships for Tesco. If their database integrity is compromised, then they will not be able to provide the service they promise to their customers.

If the enterprise is divided into many services, each of which requires DB access, there is a higher chance that the data integrity can be compromised.

Integrity rules Entity Integrity: No attribute of a Primary key must be null. When anything is added to a databse, you need to check the primary key. A Primary Key can be comprised of several attributes. If a null PK is added the db will be corrupted and need to be thrown out/recovered: expensive. Referential Integrity: Foreign keys must refer to things which actually exist. Delete is the problem here. Need to delete record from all tables which call that foreign key. Especially confusing for large and enterprises with lots of services. If a foreign key exists in a relation, then the foreign key must matchsee slides General constraints: Many here. About what data values are allowed. ie. Cant apply for skydiving if too young, cant apply for insurance if too many restrictions, no deliveries outside of the UK, etc

Where should validation be done? All three places! In the model in the view and the controller. Advantage of validating in the view: extremely good error messages, faster feedback. compare to errors thrown by the model ie. errorKeyVal etc. Each component should check. Also, then there is redundancy of data integrity checking. model programmers might not trust ux designers etc.

rails auto generates a bunch of validation methods. When something is wrong, it tells you everything wrong with it.

Rails doesnt: maintain data in the form which you got right (which is good convention).

14   Database Scalability

By now we are familiar with the three tier architecture: Presentation - Business Services - Data services.

As the enterprise scales up, we will need more more servers to provide the business tier. The internet is suited to this because the RESTful (Representational state transfer) architecture means that the server doesnt store any state, its all held by the client in the form of cookies etc. This is good because therefore load balancing is easy, redundancy of server is easy because a user isnt tied to using any server, they can go to any other server and be served in the same way.

Unfortunately, the bottleneck is then at the DB. Many servers querying the DB will lead to it being overloaded.

Databases are represented as stacks of cylinders because they are traditionally stored on discs.

Disks: spinning slow! Typically 100 000 times longer to access something from a disk than from solid state.

How to solve the DB bottleneck? Replication? We add more more servers to the business tier, why not add more DBs? Plus is that databases are now running in parallel, so any one can be read from, so reading from them is much quicker. Minus is that each database needs to be written to when new data is added to the DBs, stopping the whole system until it is dealt with, slowing it all down. Solution might be that we might not need absolute accuracy. This is something called eventual consistency: do the stack of writes at a convenient later time.

Do we allow some slackness before a change is made to the system? On something like fb and twitter, people might not like it. OTher times, there is a noticable delay (a few minutes) but you get a confirmation message. Problem might be that a request is resubmitted by a user who doesnt receive confirmation that their action has gone through. Partition by table. Faster reads because the data + use cases are partitioned. Fast writes for the same reason. Problem is when there are dependencies, like closing an account, which all databases look at. Eg. Facebook could partition its services onto different servers. This is slow when there are dependencies. Ideally, services which dont require the same data. This is unlikely though, Companies want to be seen as one entity, ie their services should be linked. Partition by service. BT can offer phone lines, internet, TV, separately. Faster reads, faster writes again - similar to partition by table. Can be frustrating if you have to keep giving your details to the same company over and over again. If you change your address for one service, it needs to be changed for the others.

Another advantage can be that you can divide your company into specialists: One team can focus solely and be experts on one service, while another team can be experts on another service.

Example: Facebook could partition its databases by service across servers (walls, profiles, authorisation, abuse, logging, ...)

DB management systems:

As far as we are concerned are likely to be row orientated management systems. Could also be column oriented management systems.

Row oriented is good for one record operations: because the data for one record is contiguous, so much quicker to access on DB disks!

Column oriented is better for doing column operations: eg. getting all columns requires retrieving all records, throwing most of the data away and keeping the one feature from each record. Much quicker if you want to sum across all records, for instance. Questions like how much did we sell last month, What was our lowest stock last month etc.

Sharding: A database management system may shard a table by storing different rows on different servers. Eg. might shard records by EU/NA. Know the query should be one or the other by ISP, GPS, location services etc if you have geographical knowledge. Eg. going to amazon.com rather than amazon uk! Your basket doesnt carry across because the services is separate. Eg. sharding by van colour would be really bad because 90% of vans are white, so one db would be overloaded.

Not only SQL = NoSQL DB management systems may replace a table with a simple key/value store. Advantage is that the data doesnt need to be structured. Can be stored Google does

this, but they can search unstructured data very well. More for a DB course.

15   Data Warehousing

Online transactional transaction processing

Beyond storing your data, you need analysis of the data you have in order to find something useful from it. Eg. its not a good idea to open a new hotel in Barcelona because we arent getting enough bookings, or conversely it is a good idea to because bookings are up etc. OLAP: Online Analytical Processing.

Where do we get the data from? Can buy it from other companies. Can use footfall information.

Makes a data warehouse.

What to do with that data? Change your business for the better. Record failed searches so you can find out what your customers are looking for but you dont stock/provide etc. Find most successful products, offer products to customers who are likely to like them.

Data Extraction: Identification and collation of the data from different sources.

Data transformation: integration, cleaning, rearrangement and consolidation. Can be difficult to do data transformation: Dr David Wakeling, Mr Wakeling, his sister is also a dr, etc.

Data Loading

Key point is that data ages over time. Very old data might be necessary. Not often stored on disks, doesnt need to be accessed frequently.

Data storage pyramid: top point is recent figures, might need to be accessed quickly, frequently. Towards the bottom of the pyramid, the wide base, the data is older, how has the shope been doing in the last decade etc. Data archives at the bottom might be stored on cd or tape.

ie. top is interactive data. Almost live data. Close to the concept of a data dashboard. Interactive sector stores unsettled transactional data, almost immediately. Information can still be changed: (Almost transactional db) a flight might be cancelled, credit may be refused, which would be noted.

Integrated sector. Might be end of the day or week. Integrated with other data settled there. Can be scrutinised.

Near Line data is older, has been dealt with, no longer needs to be changed because its been in the previous tiers already so the chances of accessing it are less, so it goes onto slower technology.

Archival sector. Data can nowadays be stored forever. The probability of being accessed though is very low. The orgs forget nothing. Banks need to store mortgages for years as well.

Retrieving data from the transactional database isnt a zero cost operation. In fact, if a website is performing badly, asking questions can make the website perform worse, driving customers worse and causing a vicious cycle. The Transactional DB is meant for customers, not for the data warehouse. Shouldnt mine your data at the expense of the user.

Need to answer the question: How are we doing right now? Last week? This year? Past 5 years? Historically. Other businesses will do this, so you need to. They will know what their customers want and they will provide it. Thats why you need a data warehouse.

16   Cloud architecture

Analogy between providing electricity and the cloud.

Advantage of this.

Rapid Elasticity. If you buy another 400 pcs, where will you put them? Enterprises need to scale very fast.

Resource Pooling: more for cloud providers. Can get customers to share machines. Can use vitual machines rather than physical. This is known as multitenancy. If a provider buys some machines, you want them to be useful work at all times. Therefore its useful to be able to load balance the virtual machines on the physical machines so they are always working. The VMs are checkpointed so if a physical machine is broken, the VM can be picked up by another physical machine without dropping service. Also, costs are low.

Measured Service.

Cloud Adoption:

Last time: Different levels of architecture: Bare machine, operating system, software as a service.

Above that, the infrastructure is either: Public or private cloud.

Public Cloud vs. Private Cloud

Triangle of CIA: Confidentiality(Only those with access rights can see the data), Integrity (IF you store a 6, you get a 6 back), Availability(When you want your data it is there)

Public clouds might offer better data assurance because:

They are hardened by continual hacking. Of all users, there are a small %age of scumbags who will attack large organisations for the notoriety. Also rival organisations and governments fund attacks. If they would have fallen prey, they would have already fallen victim to it by now and be out of business. And if they havent been fixed, they will be soon. Staffed by top people. Only large data centres can attract the best people, because they will want to work for the best/biggest/most prestigious companies. So industry leaders work for the largest companies. The best data centres can be built for these largest of companies. The largest companies do not buy standard hardware and they design their own computers and servers and buy them in bulk from China. Therefore they offer more assurance because they are built with the best security kit. Their stuff is often recent because they are also expanding and have the capital to create new data centres as they need it.

Private clouds offer less assurance because of they cannot offer the above.

Sporadic penetration testing: Companies cannot test themselves against penetration easily: you tend not to see your flaws, and you also know the intricacies of your architecture so you are inherently biased. Also, security companies arent often invited back to the same place because the company supposedly fixed the faults that the tester found. These tests are expensive, and easy to get lax on. Liable to perimeter complacency. May have been state of the art, but then becomes outmoded. Also, the people who put the system in place wont be able to foresee problems which they havent already thought of. The risk will never

have occurred to them. staffed by good, but not the best people. This may leave them open to more sophisticated attacks. So the private cloud may suffer from fewer attacks, but the staff wont be the very best. The customers and board members of a company might not want to hear that they are being served by decent but not the best staff. They dont have the buying power of the public cloud to buy the newest kit as soon as it is on the market.

Service level agreement Legal contract between you and the cloud provider.

Five nines up time is 99.999% uptime. Never promised by cloud providers, thats safety level insurance.

RESTful cloud

Data movement is still slow and expensive. But the cloud offers the ability to quickly

Edward Snowden:

At one point people thought that the big data providers wouldnt give the data away to the government. Now the cloud providers have had a rethink, Google encrypts data everywhere it sends it.

As you can see the paragraphs are separated by any number of blank lines. You can easily get a bulleted list of items with itemize:

- First item Help me!
- Second item
- Third item

A related environment is 'enumerate':

1. First item
   - First
   - Second
   - Third
2. Second item
3. Third item

The last useful one like this is 'description':

First First item

Second Second item

Last Third item

You can have different fonts *italics*, `teletype`, **bold**, sans serif, SMALL CAPS.

16.1 Mathematics

This is a new section instead of 1. You can put mathematics in the text by enclosing it in dollar signs: $x + y_i^{2 \log a} = e^\pi$. Here is an equation:

$$e^{i\pi} = -1 \tag{1}$$

Here is a displayed equation:

$$\sum_{n=1}^{\infty} \tan(\theta_n + \alpha) \gg \frac{\phi}{49} \tag{2}$$

That was in equation (2).

$$\mathcal{N} = \begin{cases} 1 & \text{if } q < 0 \\ -\pi + \phi^2 & \text{otherwise} \end{cases} \tag{3}$$

Here are vectors and matrices

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ a \end{pmatrix} \qquad \mathbf{Q} = \begin{bmatrix} 1 & 45 & 23 \\ 2 & 12 & y \end{bmatrix} \tag{4}$$

This is a bit of writing to finish the section before the paragraph.

References   The url is `http://www.dcs.ex.ac.uk/˜somebody`.

$$\sum_{n=1}^{\infty} \tan(\theta_n + \alpha) \gg \frac{\phi}{49}$$