Chapter 1

Introduction

1.1 What's it all about?

Cloud computing is essentially renting computing resources as needed to build-up systems in a quick self-service way. Think of it as booking a ryanair flight (vs buying your own plane), renting a car in a foreign country (vs buying one) or taking an AirBNB (vs buying a holiday home). Like travel planning, learning how to design, provision and utilise cloud computing resources is a learned skill developed through practice.

1.2 Backend

In this module we will primarily focus on backend processing, as opposed to frontend computing (desktops, mobile devices). As with any computing system, backend processing generally involves three key ingredients:

Compute power: in the form of CPU cycles in server hardware, possibly shared by a hypervisor and managed by an operating system.

Storage: in the form of magentic disks, SSDs, tape and other media that is directly attached to server hardware or connected over a network.

Connectivity: so that the remote computing resource can be utilised by its intended consumers.

The earliest large computers (so-called mainframes like IBM System/360) led to smaller minicomputers (DEC PDP 8/11, VAX, IBM AS/400) and later on to PC-based servers running Linux / UNIX / Windows. Yet many of the ideas we'll see in Cloud particular around pricing and pay-per-use actually were and are evident in Mainframes.

1.2.1 Batch processing

Historically a lot of backend processing was batch-oriented. This is still encountered in business processes (e.g. a bank processes payments overnight or utility sends a bill once a month).

1.2.2 Services

The alternative to batch processing is service-based processing, where a server allows one or more clients to connect and processes data in real time. Services commonly encountered include:

- **On the public internet** are web servers (HTTP, HTTPS), file transfer protocol (FTP), domain name servers (DNS), remote shell login (SSH), remote desktop (RDP), network time protocol (NTP), remote sync (rsync), virtual private network (VPN).
- **Within private networks:** all of the above plus file sharing (NFS, CIFS/SMB, AFS), database servers (MySQL, PostgreSQL, Oracle etc), message brokers (ActiveMQ, RabbitMQ), media sharing (UPnP/DLNA), block storage (iSCSI), mainframe access (3270, 5250 emulation), insecure remote shell login (telnet, rsh) and custom servers. Many of these services in theory could run over the internet but for security, performance and other reasons they generally don't.

We will assume in this module that all services communicate over standard TCP/IP. For a given service, the client and server implement a common protocol that normally has an associated standard port number.

1.2.3 Building systems

It is important to remember in practice that a service may be a client of another service, and that most systems of even modest complexity involve multiple interdependent services:

- A web application (PHP on Apache/mod_php or Spring with Tomcat) might access a database (Oracle), a messaging queue (ActiveMQ) and an in-memory cache (Redis).
- A file server running Windows Server provides SMB/CIFS file shares that are on a drive provisioned on an iSCSI volume shared from a FreeBSD UNIX machine.

1.3 Service requirements

Assuming we are dealing with backend server environments, our services are assumed to consist of software running on appropriate hardware. Server hardware requires power, connectivity and cooling in a secure and fireproof environment with appropriate channels to manage the hardware remotely. Normally we require resiliency in the infrastructural services and connectivity so that the service remains available. We can first distinguish among a number of common patterns for where our services are hosted:

On-site within a suitably equipped data centre environment providing power, connectivity, cooling and security with the appropriate resiliency for the required service availability.

Co-located where our hardware is located in a data centre facility that provides space, power, connectivity, cooling and security with the appropriate levels of resiliency.

Hosted where a provider provisions equipment and software that we require.

- Hosted environments tend to involve manual setup and can provide both bare-metal servers, managed physical servers and virtual servers.
- Hosting provider will themselves be provisioning onsite or co-lo, or may be reselling from elsewhere!
- Many co-lo data centres also offer hosting as a service.

All of these environments require significant upfront *capital expenditure*, must be right-sized from the beginning and scaling requires significant effort.

1.4 Expenditure

Setting up and running any service requires expenditure, which can be broken down into:

Capital expenditure (CAPEX): money spent on goods/services necessary for future performance. Usually fixed / non-consumable assets. CAPEX is often made using borrowed money.

Operational expenditure (OPEX): money spent on goods/services necessary for day-to-day running of the business. Often supplies/inputs into output. Needs to be self-financing if the service/business is to survive.

Note that whether a particular item is CAPEX or OPEX depends on the context. To a homeowner, a bale of concrete blocks for an extension represents CAPEX. To the shop supplying them, they are an OPEX to buy.

Key advantage of cloud computing is shifting CAPEX to OPEX.

1.5 Cloud definition and characteristics

The term “cloud” primarily appeared in its current context when AWS released their so-called Elastic Compute Cloud in 2006. Rather than an explicit definition, NIST describes five essential characteristics for a particular service to be termed a “cloud service”:

On-demand self-service allowing consumer to provision required services without human interaction.

Broad network access allowing consumer to access provisioned services over standard network interfaces (e.g. TCP/IP over internet).

Resource pooling where provider uses a large pool of resources to serve consumers on-demand, without the consumer having any affinity to a physical resource.

Rapid elasticity to enable a consumer to vertically/horizontally scale their provisioned capacity.

Measured service where a consumer pays for provisioned/used services at a relatively granular level, subject to resource limits.

If a service broadly meets these, then it can be termed a cloud service. They key differentiating factors are the self-service nature, the locatation independence, the seemingly unlimited pool of resources on the backend and the ability to control and pay for what you use on a short-term basis.

1.6 Main providers

The main cloud providers are shown in Table 1.1.

Provider	Service	
Amazon	Amazon Web Services	AWS
Microsoft	Azure	
Google	Google Compute Cloud	
IBM	IBM Cloud	
Digital Ocean		

Table 1.1: Main cloud providers

1.7 Cloud service models

Within the cloud NIST describes three different service models:

Infrastructure as a Service (Iaas): provider makes available compute power in the form of virtual machines, storage in multiple forms and allows networks to be constructed to link these resources together. Obviously you don't have access to the hardware.

- **How to identify:** you are dealing with CPU / RAM / storage specifications, operating systems (Windows / Linux), networking (IP addressing, subnet masks).

Platform as a Service (PaaS): components such as database servers, integration components (queues, notification systems), object storage that can be provisioned directly from the web / CLI.

- **How to identify:** you're not directly interacting with CPU, RAM, storage, OS, networking.

Software as a Service (SaaS): provider makes software available for use over standard network protocols (HTTP/RDP/SSH) either by humans or an API (e.g. Github).

In reality, the lines between IaaS, PaaS and SaaS are often blurred. They are probably better thought of as a continuum from SaaS through PaaS to IaaS. It can be difficult to place many cloud services exclusively into one of these three categories, nor is it necessary to. Many solutions built including cloud technologies will include all three.

1.8 Interface

Most cloud providers support a number of channels for provisioning resources and interacting with them:

Web console providing manual point-and-click interaction

- Easy to explore, BUT not a good idea to use exclusively long-term.

Command-line interface (CLI) command can be installed on your own computer and can be run from PowerShell / Bash

- It can be tempting to avoid the CLI but this is generally a foolish decision in terms of time, precision and opportunities to automate!

- The CLI commands for most providers (e.g. `aws`, `gcloud`) provide an extensive online help system with examples. For AWS try typing `aws help`.
- CLI avoids cumbersome manual login process - another reason to use it.

Software Development Kit (SDK): for applications running on cloud and elsewhere (local desktop, mobile) that need to interact with cloud resources.

- We won't spend much time with SDKs but may encounter them later on!
- Example: a Python script needs to push a message to a messaging queue provisioned on AWS SQS.

1.9 Tips

- **Ignore ill-informed hype** as much as possible. Many people commenting on cloud (and other) technological matters are no more informed that a random person on the street. Always refer to multiple sources, provider manuals and your own experience.
- Cloud vs. traditional non-cloud is **not inherently good or bad in itself**. The *business need* and *technical suitability* should always dictate if a cloud-based or onsite/co-located/hosted service is the right choice.
- Many **systems will have both cloud and non-cloud components**. Hybrid solutions where a cloud-based copmponent communicates with an onsite component often provide the optimal solution.
- There is often **more than one right answer** to a particular problem. The preferred solution might not always be the most technically "correct".
- Learn and use **Command-Line Interfaces (CLIs)** for everything you use where possible. Not just cloud infrastructure but everything!
- **Automation is key**. You NEED to be able to script repetitive operations so that you don't waste time on them, and that they're done correctly, on schedule, every time.