Corndel DevOps Engineering Programme

in association with Softwire

Module 11: Tooling & Cloud Infrastructure

Corndel Digital.

DB86

Introduction

References to **cloud computing** are commonplace these days, but what exactly is **the cloud**? Definitions can vary, but for our purposes we will consider any scenario in which a user accesses digital resources, virtualised hardware, or software *on demand* over the internet as being an instance of cloud computing. The "on demand" caveat is important: accessing real metal servers in a data centre is generally **not** considered cloud computing.

One fundamental advantage of cloud computing is the flexibility it has introduced, especially around non-traditional services. While services like **messaging systems**, **serverless computing**, etc. can all exist in traditional data centres, cloud computing has made these services more accessible to a wider variety of users.

Table of Contents

Chapter 1

Types of Cloud

Types of Cloud Services

Cloud computing services can be broken down into two broad categories: cloud infrastructure and cloud end-user software.

Cloud Infrastructure

- **Infrastructure as a Service (IaaS)** is the provision of virtual hardware. Here the service provider makes available virtualised hardware resources in the form of virtual machines (VMs), storage, etc. Clients rent these and use them in place of "on premises" computing hardware.
- **Platform as a Service (PaaS)** is the provision of virtualised hardware running some software in addition to the basic IaaS provision (typically software to enable building and maintaining a cloud system), but not end-user software. Examples include remote servers running operating systems, client management interfaces for public cloud resources, databases and so on.

Cloud end-user software

End-user software (i.e. full software applications) can be provided through the cloud in a number of different models (we will return to these in more detail later in the module):

- **Software as a Service (SaaS)** - commoditised software applications provided online for free or through subscriptions - products like Google Docs, Slack and Dropbox;
- **Enterprise** software licensed to big businesses or governments;
- **Bespoke** software written as a one-off solution for a specific client.

Types of Cloud Provider

There are a number of different approaches for setting up cloud computing.

Public Cloud

The best known cloud model is the **public cloud**. This is where a third-party provider makes infrastructure or services available to the public via the internet.

Key aspects of a public cloud are:

1. A third-party service provider makes the computing resources or services available to the public online;
2. The service is multi-tenanted, i.e it is shared between multiple users and clients. However there is generally a strict separation of the data and activities of one client from another;
3. It is the third-party provider who maintains the physical hardware and network, and provides tools and mechanisms to allocate resources to individual clients. This can include client-facing self-service interfaces and virtualisation software.

Often clients will purchase cloud services on a pay-as-you-go (PAYG) basis, for example a monthly per-seat subscription for software services or an hourly rate for virtual machines. However other models are available.

Important advantages of using a public cloud are:

- Clients have no hardware maintenance costs;
- Provides smaller organisations with access to technologies and tools which would otherwise be beyond their reach (possible because the costs are effectively shared between clients);
- Clients can scale their cloud for the distant foreseeable future since the resources available to third-party providers far exceed those needed by most organisations;
- Scaling is on-demand and normally client-managed making it fast and comparatively simple;
- Since the provider's resources are shared between multiple clients it allows hardware to be used very efficiently. This translates into lower costs for clients compared to maintaining dedicated hardware;
- When well architected, offers a secure, resilient, widely available solution.

Set against these advantages, it should be noted that there are some disadvantages. In the case of low-level cloud services (infrastructure or self-built software based on cloud platforms), clients typically need access to cloud experts to create and maintain their cloud ecosystem. This is a cost which needs to be factored into any cloud deployment.

At a hardware level, public cloud architecture normally relies on virtualisation to segregate different clients' data and activities. This may not satisfy regulatory compliance in some industries. Similarly, at a software level, clients are dependent on the provider to ensure that data leaks do not occur between users. In general, however, security issues in the platform technology itself are often very unlikely. At the infrastructure and platform levels, security problems are more likely to result from incorrect configuration on the client side rather than an inherent weakness with virtualisation or other core technologies.

Vendor lock-in is a very real concern. While different cloud providers offer services which are similar in functionality, moving from one to another requires a great deal of expertise. Clients' cloud setups are dependent on the services of a particular provider and it subsequently becomes very difficult to move to another provider. Migrating from a local IT infrastructure to a cloud-based one can be costly and requires careful planning. This is a specialist area in and of itself and will not be covered in detail here.

Public Cloud Providers

Many different companies sell cloud infrastructure services. However, the worldwide market is dominated by three main players:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure

Of these, AWS is by far the largest, typically credited with holding nearly a third of the total market. Azure comes in second with around 20% market share, and GCP around 10%. The remainder is made up of a long tail of smaller providers. We will generally concentrate on the "big three" in this module.

Private Cloud

A private cloud is the provision of remote computing infrastructure dedicated to a single organisation.

Hardware and data centres may be owned and managed by the organisation itself, or a third-party provider may be contracted to provide them as a service. In the latter case the third party may operate the cloud in a shared environment alongside other clouds, but the organisation's infrastructure will be isolated from others either logically or physically.

This gives rise to a number of approaches for provisioning private clouds.

Virtual: the organisation's cloud is provisioned in a third-party data centre, running on shared hardware - i.e. the hardware may also host the cloud infrastructure for other organisations. However, software virtualisation maintains a strict segregation between clients. This approach differs little from provisioning via a public cloud provider.

Managed: the organisation's cloud runs in a third-party data centre. The data centre as a whole provides cloud services for multiple clients. However each client's cloud runs on dedicated hardware to keep them physically separated. The third-party service provider runs the cloud on behalf of the organisation.

Hosted: similar to managed provision, but the cloud is managed by the organisation's own staff.

In-house / internal: the organisation builds and runs their own data centres. This is the most secure approach, but also the most costly. It is only really suited to enterprises or governments whose large user base make it an economic investment.

Private clouds offer various advantages:

1. Ability to develop proprietary architecture which might not be available through an off-the-shelf public cloud;
2. Offers the organisation complete control over their cloud;
3. Resources dedicated to the organisation which may improve performance;
4. Single-tenant environment, potentially increasing security;
5. May allow regulatory compliance to be achieved more easily than other cloud solutions.

Set against this are a number of disadvantages:

- Generally more expensive than public clouds;
- Depending on the provisioning model, private clouds can be complex to set up and require specialist staff to maintain;

Hybrid Cloud

A hybrid cloud is an organisational cloud which includes a mixture of public, private and sometimes on-premises elements. This might be a desirable approach in situations where demand for computing resources fluctuates significantly over comparatively short periods of time, or if there are differing security and access requirements for different parts of the system.

In these examples, the private cloud would provide for the base load and support high-security activities with strict access restrictions. The public cloud would allow the system to remain available during times of peak demand (or provide resources for occasional high-capacity activities such as data processing), and also support much more open access to less secure functionality.

Hybrid clouds can also be used to gradually migrate IT capabilities into a cloud environment without massive internal upheaval. Similarly they allow new IT functions to be added in the cloud without requiring existing internal infrastructure to be thrown away.

A hybrid approach can use networking technologies such as VPNs to create the appearance of being a single homogeneous system from the end-user's perspective.

Multi-Cloud

Often, organisations will limit themselves to a single cloud provider (or at most individual providers for their on-premises, private and public cloud services). An alternative approach is a multi-cloud in which individual cloud services are used from multiple cloud providers. The main reasons for this are:

- To add redundancy and improve resilience by eliminating reliance on a single cloud provider;
- To pick-n-mix the best services available on the market from different providers.

Multi-cloud clearly has similarities with hybrid clouds, the key distinction being the potentially large number of different providers in each arena.

It potentially offers the best of all worlds. However, individual cloud providers normally have services which work well together, whereas trying to pair services from different providers can result in unexpected incompatibilities, as well as resiliency and security issues.

Community Cloud

Clients from similar backgrounds, such as industry sectors, will normally have similar requirements from a cloud service. This may be in terms of the features they require, or with regards to security, privacy, regulatory compliance, etc.

A community cloud is a public cloud service which is specifically designed to cater to one such group (aka community). It is a multi-tenanted environment designed specifically to be used by members of the community.

Government Specific

Very similar to the community cloud concept are government-specific clouds. Governments tend to have substantial IT infrastructure which needs to have high levels of security. This has led to a variety of cloud offerings specifically targeted to government use.

For example, the UK Government runs GOV.UK PaaS for hosting public sector services. Under the surface, this runs on AWS.

Chapter 2

Services Offered

Cloud infrastructure providers offer a huge range of different services. As an example, as of August 2020 Amazon Web Services (AWS) has in the region of 200 individually identifiable services on offer, ranging from raw compute and storage to cloud management tools and more esoteric services such as quantum computing and satellite communications. Even smaller providers with simpler offerings typically have tens of service types allowing both the creation and management of cloud services.

This section provides a broad overview of the types of IaaS & PaaS services available from the main public cloud infrastructure providers (AWS, Azure and GCP). We will cover a subset of these in more detail later in the course.

Core Services

There are two core areas which are fundamental to cloud infrastructure provision: computation and storage.

Computation

No IT infrastructure can function without raw computer processing power. Compute services offer the ability to run code, either on Virtual Machines (VMs) or in serverless environments.

Virtual Machines

Amazon's Elastic Cloud Compute (EC2) service, Azure's Virtual Machines and Google's Compute Engine all provide VMs running in a cloud environment. These are the backbone of all cloud computing services, as many other cloud products provision virtual machine capacity in the background.

Clients may determine the amount of processing power and memory available to each machine they set up, as well as which operating system it runs. They can also allocate arbitrary amounts of storage on one or more virtual disks.

Cloud VMs may be used in any way the client wishes, for example often as a web server or to carry out data processing, and can be interacted with via Secure Shell (SSH) or Remote Desktop Protocol (RDP).

There are various pricing models for Virtual Machines, of which AWS provides a representative set:

- **On-demand**: the client pays for each provisioned VM according to how long they use it for;
- **Reserved instances**: the client commits to paying for the machine for a long period of time (for example a year or more) in exchange for substantially reduced costs over the on-demand model;
- **Spot instances**: cloud service providers sell off spare compute capacity at what can be a fraction of the normal on-demand price. Clients can make large savings this way. The caveat is that the price fluctuates over time depending on the amount of spare capacity available. Clients set a maximum price they are prepared to pay, and the VM turns off if the real-time price rises above that level. Consequently spot instances are best suited for non mission-critical tasks which can be deferred if needs be (for example, out of hours data processing).

Storage is usually billed separately from the compute resource itself, for example on a Gigabyte per month basis.

Serverless Environments

"Traditionally", software in the cloud has required an always-on server dedicated to a particular client. **Serverless computing** is a more recent development which provides an alternative model. It has mainly gained traction since the early 2010s.

In the serverless approach, there is no dedicated server or VM running a piece of software. Instead, when an event occurs which the software should deal with, the cloud service provider immediately allocates computing resources to the software which carries out the necessary task. For example, the software might be a web application, and the event an HTTP request. When the request occurs, the software is run and returns the appropriate response.

All the major cloud providers have serverless services, such as AWS Lambda, Azure Functions and Google Cloud Functions.

The principal advantage of serverless environments is that because compute resource is only made available to clients in the exact moment it is needed, the service provider can be extremely efficient with resource usage. This can result in significant cost savings to the client since they are only charged for the exact amount of resource consumed by their application.

In addition, a lot of the pain of handling uneven demand is removed from the client since the service automatically allocates compute resources, including multiple instances of the same application, as needed (within limits). This is sometimes termed **elastic** capacity since the amount of resource shrinks and grows moment-by-moment as necessary (as opposed to **scalable** architectures which generally have at least one server running at all times, and add more servers dynamically as required).

Unfortunately, going serverless isn't a panacea for all cloud compute scenarios. It has a number of disadvantages:

1. It is more complex to set up a serverless environment than a VM. As a minimum, some kind of event detector layer is needed, which generally results in additional cloud services needing to be set up;
2. Infrequently used serverless applications can have high response times (latency) because their code needs to be loaded into the runtime environment when the trigger event occurs. There are ways around this, but again such approaches increase the complexity of the solution;
3. For applications where demand is very high and / or continuous, particularly where demand exceeds the capacity of one or more servers, serverless environments will be more expensive than dedicated Virtual Machines. It is also worth noting that cloud providers often place limitations on the maximum level of usage, so extremely high capacity services simply may not be possible;
4. Since serverless applications are dependent on a number of specialised services, vendor lock-in becomes more of a problem since in order to move to a different provider it is necessary to recreate the entire environment, not just a VM.

Cloud providers normally provide an ecosystem of tooling and services to support the setup and monitoring of serverless compute capacity.

Containerised Services

As we learnt earlier in the course, containers are self-contained packages containing everything needed to run a specific application.

It is fast to spin up or destroy containers, allowing containerised infrastructures to scale quickly in response to demand changes. Cloud computing is well-suited to scaling containerised applications.

However, there is complexity in containerised infrastructure: containers need to be assigned access to host resources such as networking and storage space, without exceeding the capacity of the host. This is particularly problematic in auto-scaling environments.

Container orchestration platforms are designed to manage this complexity. These automate the entire process, ensuring that an appropriate number of containers are made available at any time, and managing the connections between containers and their access to host resources.

The most widely used container orchestration platform is **Kubernetes**. This is cloud-agnostic, but the major cloud providers all provide their own managed Kubernetes service: AWS Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS) and Google Kubernetes Engine (GKE).

In addition, some cloud providers also offer their own proprietary container orchestration or runtime environments: AWS Elastic Container Service (ECS), Azure Container Instances and Google Cloud Run.

> *We'll cover Kubernetes and container orchestration later in the course.*

Storage

Databases

Cloud providers generally offer a range of databases which may be provisioned in a similar way to compute capacity. The vast majority of the commonly used database technologies have one or more corresponding cloud services.

Other forms of storage

Besides databases, which are mainly useful for dynamic data, there exist a range of other storage options which handle files.

Blob storage is designed to store large volumes of infrequently modified data. Data is stored in a flat, distributed structure, which may have significant amounts of arbitrary metadata attached to it. Objects in blob storage may be immutable, so that "modifying" an object actually creates a brand new replacement copy. Consequently it can be comparatively slow to write to these services. However, they lend themselves to remote access at scale, and can have a Content Distribution Network (CDN) placed in front of them which makes them very good for making files available online. Examples include Amazon Simple Storage Service (S3), Azure Blob Storage and Google Cloud Storage.

> ***Blob** is an acronym for **binary large object**. These are a natural place to store binary files like images, audio and video which are generally accessed in their entirety or not at all.*

File storage similar to Network-Attached Storage (NAS), i.e. storage which can be addressed as a disk is made available remotely. It has all the characteristics of a local disk such as directory structures and limited metadata which describes only the characteristics of the file. It is good for high-performance applications, or where the storage needs to be mounted for local access. However it cannot scale indefinitely in terms of numbers of files, volume of stored data or throughput. Examples include Amazon Elastic File System (EFS), Azure Files and Google Filestore.

Block (or **disk**) **storage**, which provides disk-like storage. It is performant and often used for the drives in Virtual Machines. Examples include Amazon Elastic Block Store (EBS), Azure Disk Storage and Google Persistent Disk.

Archive storage services take stored data offline. They are intended to allow long-term, highly resilient data storage at the expense of extremely slow data access times, and are only suited for situations where data is needed only rarely, if ever. Data may be stored on highly durable physical media such as tapes, in one or more secure locations. If the client needs access to the data they may need to submit a request to the service provider in advance who will load it from the media onto another remotely accessible storage service.

> A good place for putting things you *probably* won't need to use: back-ups, audit logs, records that need to be kept for regulatory compliance but don't actually need to access.

Cloud providers sometimes apply multiple tiers to any given storage solution:

1. **Hot storage** is that which is always available and can be accessed quickly. It has higher storage costs but lower access costs.
2. Conversely **cool** or **cold storage** has higher latency (ranging from seconds to minutes or even hours). It has higher access costs but lower storage costs.

Good storage design will take account of this hierarchy and place data in the most cost-efficient location.

As with all cloud services, access to storage can be restricted to specific users, roles or by other criteria. In some services it is also possible to place files into a read-only state so that they cannot be accidentally (or maliciously) modified or removed.

Other Services

Outside these core services, there is a range of other cloud services available.

Messaging

Many systems will require transfer of data and events between different parts (e.g. in a microservices architecture). Cloud providers offer messaging technologies to handle this, generally falling into one of two models:

Message queues, in which messages are placed into an asynchronous queue by a software service (the producer), and read out of the queue by another service (a consumer) which acts on it. The consumer polls the queue on a regular basis to see if there are messages which need actioning. Messages remain in the queue until they are read, at which point they are removed. This means that although there may be multiple producers and consumers each message will normally only be processed once (although depending on the queue implementation this isn't always guaranteed). It should also be noted that message ordering is also not guaranteed except in First In First Out (FIFO) queues. This approach may be referred to as **one-to-one** or **point-to-point**. Examples of queueing services include AWS Simple Queue Service (SQS), Azure Storage Queues and some configurations of GCP Pub / Sub.

Message queues are particularly useful for processes where demand can be lumpy but individual messages don't need to be processed immediately. They can provide a buffer between a public facing website and a backend system, preventing the latter from being overwhelmed. For example, there is often a surge in people registering to vote before elections; putting these requests onto queues can ensure every user can submit their registration by the deadline, even if those requests don't get processed immediately.

Publish / Subscribe notification services differ from message queues in that messages are published to topics. Each topic has registered subscribers, all of whom receive every message which is published to the topic. Unlike a message queue, in this model messages are pushed to subscribers rather than consumers pulling messages on demand. As with queues, depending on the service implementation, messages may occasionally be received more than once and message ordering may not be guaranteed. Examples of this type of service include AWS Simple Notification Service (SNS), Azure Service Bus and some configurations of GCP Pub / Sub.

An e-commerce site might use this model: every time an order is received it generates a message on the `order received` *topic. This is subscribe to by multiple consumers: one service to send the user a confirmation email; another to update the warehouse inventory system; and another to update the accounting systems. If we want to add a new system to send a confirmation text message then this can easily be added* without disrupting any other systems.

Both these types of service are normally charged on a pay-as-you-go basis, with the price being dependent on the number of messages published and consumed, as well as the overall amount of data which gets sent.

It is also worth mentioning **real-time streams** which are designed to handle large volumes of data, for example AWS Kinesis. These deal with continuous data streams rather than discrete messages.

Monitoring, Logging & Alerting

All but the simplest software and architectures will encounter difficulties at one time or another. Bugs may become apparent, or parts of the system could become unavailable. In order to identify issues, and help resolve them, monitoring and logging tools are indispensable.

1. **Monitoring** is the act of observing and reporting on the current (and historical) state of the cloud system's components. It allows near real-time identification of issues such as individual elements of the system going offline, or over utilisation of some aspect of the service. Automated responses to problems may be built in to the monitoring so as to allow self-healing or scaling of the service as needed.
2. **Logging** is the creation of human or machine-readable streams of data which provide information about what components of the system were doing at any time. Logs are normally stored in text files or a database and contain timestamped statements which describe a particular event which took place. For example, the log might show that a component was restarted, or that an HTTP request was received, or a particular data processing script was initialised. Logs are used by developers and system engineers to understand the sequence of events which took place over a given time period, and can be very helpful for identifying the cause of a problem which occurred.
3. In addition, **alarms** or **alerts** may be set up. These are where the engineer identifies thresholds for specific metrics (e.g. error rates or response times) which, if crossed, will cause a notification to be sent out. This can be combined with automated actions which the monitoring service should take in the event of an alarm being triggered.

Cloud providers generally offer a one-stop monitoring service which pulls together information into a single location. Users may configure what information is available by identifying specific services which should be observed and which metrics should be reported. The resulting data is displayed in convenient tabular or graphical format on a dashboard. Examples of these kinds of monitoring service include AWS CloudWatch, Azure Monitor and Google Cloud Platform's Monitoring service.

Some cloud providers offer services which allow highly detailed monitoring and logging of applications themselves. An example is Azure's Application Insights. For this to work, a small additional software library is installed as part of the application codebase. This library makes calls to Azure's logging services which allow it to track application activity. Combined with information which Azure has about the hosting environment this makes it possible to capture large amounts of telemetry which, used properly, enable a comprehensive understanding of the state and performance of the application and supporting cloud ecosystem.

Outside of code and system architecture, cloud providers make available a separate set of services which allow account governance, usage and cost tracking, and identification of cost efficiencies. These are beyond the scope of this discussion.

> *The methods, tooling and services for supporting monitoring and logging are wide ranging and will be addressed specifically later in the course.*

Data Pipelines

A data pipeline is a series of services which act on a set of data. The output from one service forms the input of the next one.

Broadly speaking, there are two types of data pipeline:

1. **Real-time** - in which data is streamed from a source for immediate processing, use or storage;
2. **Batch** - where data is generated over time but only ingested into the pipeline on a scheduled basis. This may be referred to as an ETL (Extract, Transform, Load) system.

Cloud providers often offer mechanisms for pulling data out of their services and distributing it to a separate storage or processing service using either model.

The breadth of tools and services in this sphere is extremely wide, each offering slightly different use cases and scale of operation. The following aims to give a flavour of the kinds of things which are possible.

Real-time / Streaming Data Pipelines

We have already discussed AWS Kinesis in the context of messaging. However another good (indeed, arguably more appropriate) use for it is real-time data pipelining. Kinesis can integrate directly with a variety of other AWS cloud offerings and stream the data they generate to other AWS services. Azure has a similar service in Azure Event Hubs.

These services can also be fed directly into transformation and analysis services such as Kinesis Data Analytics or Azure Stream Analytics and from there into storage or dashboards for visual review.

Batch Pipelines

An **ETL** (Extract, Transform, Load) process involves periodically extracting data from one or more sources, transforming it into a standard format and then loading it into storage such as a database or files. Services such as AWS Data Pipeline, Azure Data Factory and GCP Cloud Data Fusion cater to this market.

You may also come across ELT (Extract, Load, Transform) in which a pipeline moves data directly from the source into storage and then performs any necessary data transformations in situ.

Batch processing is well suited to handling large volumes of data, but due to its scheduled nature, some latency is introduced between the data first being generated and the processed output becoming available to the end-user. This is commonly used in reporting services, where transforming the data into a useable format might be relatively slow so you wouldn't want to do that process every time any of the inputs changed.

> *Consider a data aggregation service, like the UK Government's Coronavirus dashboard. Updates on cases, hospitalisations, and vaccinations will come in piecemeal from a variety of different sources and be stored in a central database. Then, once a day, a batch process will* ***extract*** *the data required to drive the dashboard,* ***transform*** *it into a more performant format, and then* ***load*** *it into the data store that drives the dashboard.*

Data Processing Tools

Alongside tools which transfer data around their ecosystems, cloud providers also offer services which carry out actual data processing. These are often based on industry standard data analytics and processing frameworks, for example Apache Spark and Apache Hadoop. Different tools are better suited to streaming and batch processing operations.

To increase throughput, large scale data processing generally makes use of clusters of computers working in parallel. Managed services such as AWS Elastic MapReduce (EMR) provide a one-stop-shop making available both the software frameworks themselves and the hardware needed to run them. Similarly, GCP has Dataproc and Azure offers HDInsight. All are able to host a range of data processing frameworks.

On a smaller scale, serverless services such as AWS Glue can automate some of the data preparation out of the box. Glue will discover and interrogate structured data in other AWS services and perform ETL operations on it. Glue is billed as a managed ETL service which is convenient and fully integrated with other parts of the AWS ecosystem. However it is not able to support very large volumes of data in the way that hosted software and full-scale data clusters available through EMR can do.

Private Package Repositories

It is common for code to make use of third-party packages which are installed using package managers such as npm (JavaScript, Node), pip (Python), NuGet (Microsoft Ecosystem, .Net, C++), Composer (PHP) and others. All of these access public repositories containing large numbers of normally open-source libraries which can be added to software during development.

It is also possible for developers to create private packages which conform to public package format but which are not publicly available. Some repositories offer paid cloud services to store private packages, such as NPM (where paying users can apply user or organisation-level scoping to the package). Alternatively, packages can be stored privately on source control services such as GitHub or Bitbucket (each of whom offer free hosting, but have paid tiers to support larger development teams).

The major public cloud providers also offer source control repository services, often supporting account-level user access control and direct integration with other services in their portfolios. Examples include AWS CodeCommit, Azure Repos and GCP Cloud Source Repositories.

AWS also offers CodeArtifact - a "managed artifact repository" which can be used to control access to packages from outside an organisation. It ensures that developers are only able to include trusted packages in their software.

Regionalisation & Availability Zones

Cloud providers almost invariably make their services available on a regional basis. This increases resilience, improves performance, and can be useful for regulatory reasons.

For example, online services made available to EU citizens need to comply with strict data privacy legislation which generally means that EU citizens' personal data can only be stored on servers located within the EU. The regional provision of services means that it is easy to adhere to this requirement. Moreover, better performance (in terms of reduced latency and potentially increased bandwidth) is achieved by keeping cloud servers comparatively close to their users.

Each region typically has multiple **availability zones**. An availability zone is an independent set of hardware infrastructure designed to be completely isolated from other availability zones in a region. Although it is not necessary to make use of multiple availability zones when developing a cloud-based architecture, doing so can provide resilience against an infrastructure failure in one zone. A highly resilient cloud system will have duplicated systems in multiple availability zones as redundancy in case one zone goes down.

In some cases it is helpful to distribute on a wider scale rather than concentrating availability of a service in a single region. A good example of this is found in **Content Delivery Network** (CDN) services. These allow resources which are stored in a particular location to be made available with low latency and high bandwidth on a much wider geographical basis. For example, in the case of AWS, some static resources might be stored in an S3 bucket. The CloudFront CDN can be configured to load data from S3, caching it in regions selected by the developer, after which the content can be downloaded at high speed by users.

Networking Services

Cloud services all rely on a sophisticated networking infrastructure, both internally (communicating between individual services) and externally (public internet facing, providing users with access to the systems which have been set up).

It is possible to make use of public cloud providers for quite some time before becoming embroiled in the minutiae of network configuration, as a lot of services set it up automatically out-of-the-box. However, it is important to realise that it exists, and that it is customisable to suit individual needs.

Chapter 3

IaaS vs PaaS

We have already briefly touched on the concepts of Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). In this section we take a more detailed look at these models for delivering cloud services.

IaaS

Infrastructure as a Service is the provision of "bare metal" resources in the cloud. It is where the cloud provider makes raw hardware available to the end user, who then takes responsibility for setting it up and using it as they see fit.

Typically, IaaS services cover the provision of servers, storage, networking infrastructure, along with virtualised instances of them. Management of physical hardware and virtualisation is handled by the cloud provider whereas everything else is handled by the user.

> *Virtualised instances of hardware are very common, as they more efficiently use the underlying physical hardware.*

Examples of IaaS services include:

1. AWS Elastic Compute Cloud (EC2), Elastic Block Store (EBS) and Simple Storage Service (S3);
2. Azure Virtual Machines, Managed Disks and Blob Storage;
3. GCP Compute and Storage

Provisioning and managing these services is normally carried out by users themselves via self-service tooling. Consequently, by extension, the IaaS product can reasonably be considered to include provisioning tools (normally web-based consoles), billing and monitoring services, logging systems, load balancing and (auto-)scaling mechanisms, and backup and reinstatement tools. I.e. the IaaS service as a whole includes everything needed to ensure that end users can get the resources they need and keep everything running reliably.

There are various reasons for adopting IaaS:

- The management of physical hardware and virtualisation is handled by the cloud provider. This removes the need to maintain on-premises hardware or data centres, and can result in significant cost savings;
- At the same time, clients retain complete control over the amount and type of underlying infrastructure they use;
- Clients retain complete control over their cloud platform's architecture and the software it runs;
- IaaS is cheaper than dedicated (owned) infrastructure for some workloads. For example for uneven or occasional use it makes little sense to keep servers permanently on standby. Its PAYG payment model means that costs can be switched on and off as demand necessitates;
- In a similar vein, IaaS can be more flexible than owned infrastructure since it is easy to modify the amount of infrastructure you have in the service - IaaS is designed to scale easily and cheaply;
- The supporting services which accompany IaaS provide an off-the-shelf solution for creating resilient solutions which are easily backed up and recovered. Again, this can save costs over a home-brew solution.

There are some reasons why it may not be appropriate to adopt IaaS:

- As time goes by, cloud providers retire older platforms. This can create legacy compatibility issues, for example if VMs are no longer available which will support a particular OS or language. In these circumstances another solution needs to be found;
- Due to the prevalence of virtualisation in the provision of IaaS, it is normally a shared-tenancy environment. This can result in security or performance concerns, although in reality these tend to be fairly minimal.

PaaS

Platform as a Service (PaaS) is an evolution of IaaS in which the cloud service provider makes available everything needed for running software applications except the actual application code and data. The intention is to create an environment into which developers can drop code and have all of the practical aspects of hosting and delivery taken care of.

PaaS includes everything available in IaaS (although details of the infrastructure can be obfuscated behind higher level services), the provision of Operating Systems and runtime environments, middleware (data transfer layers) and database services.

PaaS also commonly includes development tools such as online Integrated Development Environments (IDE) which allow developers to build software directly within the the platform. Generally, using these these is optional and they can also be regarded as a form of Software as a Service (SaaS - see next section).

There are dedicated PaaS providers such as Heroku. The major public cloud providers have PaaS offerings made up of various services, including:

- AWS Elastic Beanstalk
- Azure App Service
- Google App Engine

A PaaS case-study on 'Heroku' is included in the additional reading material for this module.

Reasons for adopting a PaaS-based approach include:

1. It can greatly increase the speed in which the hosting platform and supporting infrastructure can be set up. There is no need to build a bespoke environment - it is only necessary to concentrate on the application itself;
2. PaaS usually comes in the form of a single environment, formed of multiple components with guaranteed interoperability. There should be few challenges getting everything to work together;
3. The user has no responsibility for the underlying infrastructure. If something breaks, the vendor fixes it. Cost of ownership is therefore considerably reduced (and as with IaaS, the vendor can use underlying hardware resources efficiently by splitting capacity between clients, further reducing cost to the end user);
4. Scaling is normally automatic, within configured limits;
5. PaaS abstracts the underlying hardware, operating systems and other services, making it easier to work with them (sometimes at the cost of configurability);
6. PAYG payment models ensure users only pay for what they use, reducing costs.

However, there are some reasons for caution when adopting PaaS:

- It carries a high level of vendor lock-in / dependency. PaaS systems are vendor-specific so switching to an alternative provider necessitates a full rebuild of an application's architectural stack;
- The same security and compliance challenges exist in PaaS as with IaaS. Indeed, in many ways reliance on the cloud provider increases since data and code is shared directly with them, and their procedures and security are harder to review. This can make ensuring compliance with regulations quite difficult to achieve.

It should be noted that IaaS can quickly stray into the territory of PaaS. For example, a Virtual Machine is an IaaS offering. However, the operating system which it runs is arguably PaaS. It is difficult to provision a VM without an OS which creates a grey area between the two service models.

FaaS

Function as a Service (FaaS) is a form of PaaS which is oriented towards serverless scenarios. In FaaS, rather than deploying applications to servers (as in PaaS), raw code is uploaded to the platform itself. This is run on demand in response to trigger events.

The FaaS model has a few advantages over PaaS:

- Scaling is extremely easy - the platform is designed to add more running functions as needed;
- Billing is very precise, since users only pay for the time and resources consumed when the code runs.

Set against this are a few disadvantages:

- Applications must be designed to be stateless, and support instances running simultaneously in parallel to one another. This is a different way of thinking from server-based code;

- There are a few gotchas such as warmup time when the function has not been used for a while (the time taken for the service to load the code, build it and set up its runtime environment);
- FaaS tends to have a higher level of complexity than other solutions, making maintenance and debugging more difficult.

Examples of FaaS include:

- AWS Lambda
- Azure Functions
- Google Cloud Functions

For a more detailed discussion, see "Serverless Environments" earlier in this module.

Control vs Responsibility

In the next section we will talk about Software as a Service (SaaS) and other options for delivering online applications.

With this in mind, it should be noted that all of these approaches lie on a spectrum of control and responsibility. In a self-hosted environment, clients have complete control over every aspect of the system. However, they also have responsibility for maintaining all parts of it.

Adopting IaaS relieves clients of the responsibility for supporting hardware, but also relinquishes some control over the system. This shift continues through PaaS, and finally into SaaS where the client becomes simply a paying customer of a service provided entirely by a third party.

Chapter 4

SaaS vs Bespoke vs Enterprise

Software can be provided via the cloud, or make use of cloud functionality. There are a number of models by which this is achieved - Software as a Service (SaaS), enterprise applications and bespoke software.

SaaS

Software as a Service (SaaS) is the provision of full software applications online. The entirety of the application is hosted in the cloud, including its user interface, business logic and storage (although some allow data to be saved locally as well). Users access the application using a client such as a web browser.

SaaS can include free consumer offerings through to paid-for enterprise-level software. Often services have basic free tiers and more feature-complete paid-for services. When paid for, SaaS is normally a subscription service with monthly per-user payments - it is effectively rented software.

Specific examples of SaaS include Outlook, Gmail, Google Docs, Dropbox's web-based interface, Salesforce, Microsoft 365, LucidChart, Slack and GitHub.

The defining aspect of SaaS which differentiates it from the other options we discuss in this section is that the service is rented as-is. Its features and functionality are defined by the provider who offers it as a service off-the-shelf. This generally allows the service to be provided at a comparatively low cost, but potentially reduces the flexibility for customisation.

There are various reasons why SaaS can be a good option:

1. SaaS has a very low barrier to entry - clients simply sign up, pay some money and start using it. This is almost immediate, and the provision of free / trial tiers mean it is easy to experiment with different products before committing to one;
2. Users don't need to worry about any aspects of maintenance. Bug fixes and upgrades happen automatically, and there is no local installation to deal with. In a similar vein, issues such as security are handled by the software vendor;
3. It is a relatively low cost way to access sophisticated applications which might otherwise be out of reach either due to licensing costs or because hardware for the application is specialised and expensive;

4. SaaS is highly scalable, with costs rising in line with usage (number of users). Users can be added at will, and it is also often possible to switch on additional functionality on demand (at a price);
5. The software can be accessed anywhere there is an internet connection. This means users can be mobile, tied to neither a specific location nor a specific device.

As with any solution, there are some disadvantages:

- The software must be used as-is, so cannot easily be tailored to specific individual or organisational needs. That said, most platforms offer some organisation-level configuration, and a large customer may be able to negotiate with the vendor (either in terms of having the development of particular new features fast-tracked, or around customisations);
- SaaS is almost invariably a multi-tenant solution. This can bring with it security, compliance and performance concerns;
- Clients have no direct access to underlying data stores, which may be problematic in some scenarios;
- Upgrading from a free or basic tier to levels with higher "Enterprise" functionality can be expensive (particularly given by the time you need these features you can have a fairly large user base - this creates a significant step in costs);
- The software can become unavailable due to service outages. This may be planned (maintenance) or unplanned (infrastructure-level breakdown or some other technical issue);
- Functionality can change in unexpected or undesirable ways - the vendor controls its features and roadmap and most clients don't have any say over how these unfold;
- Online software has technical challenges associated with it, such as its requirement for a solid internet connection, and potential latency issues. These can make it inappropriate for certain situations (or at least necessitate an offline mode);
- There is a risk associated with having data and software in the hands of a third party, particularly with regards to the possibility that the vendor could go out of business, rendering data inaccessible and the software useless.

Enterprise Applications

Enterprise software can be considered a cloud solution because it is accessed online or via a network. However, it is installed on the client organisation's own servers.

In contrast with SaaS, where the software is rented as a commodity product, vendors of Enterprise applications tend to deal with small numbers of high-volume transactions. Enterprise software is licensed to clients, with a single license potentially costing large amounts of money.

In keeping with its name, Enterprise applications are typically employed by large enterprises (big businesses or governmental organisations). It is generally too expensive for smaller organisations, and offers more functionality (and associated complexity) than they need. SaaS is better suited for mass market consumption where there is a well-defined task to be performed.

While some Enterprise software can be bought off-the-shelf and used as-is, it is typical for it to be customised to the client organisation's specific requirements. To this extent, it is not uncommon for the software to offer only part of the necessary functionality at the time it is ordered. The core platform is then further developed by the provider before installation with the client.

Enterprise software vendors may be prepared to offer this level of customisation because the client effectively owns their instance of the software. The software runs on stand-alone hardware and changes will not interfere with other customers. This means many Enterprise software solutions can be made to meet the exact needs of their users. However, such flexibility has pitfalls. In particular, extensive changes may make it impossible for the vendor to provide long term support for the platform, or to apply future upgrades from their core product.

As a result, Enterprise software tends to be quite slow to commission and have a long lead time. Its high licensing cost means businesses tend to consider purchasing it very carefully. Once bought, there are numerous tasks which need to be completed before the system can be used, potentially including provision of hardware infrastructure, customisation and configuration work, testing and training (which can be onerous due to the complexity of the software).

Much of this work may be carried out by the vendor. In some instances a turn-key solution may be achievable (i.e. the vendor turns up one day, installs the software and everything is ready to go), but often additional specialist support is required (particularly with regards to hardware setup).

From a technical perspective, Enterprise solutions overcome some of the potential disadvantages of SaaS:

- With a locally-hosted, single-tenant system, the client has control over security, performance, resilience and regulatory compliance;
- There are no users outside the client organisation, allowing greater freedom to modify aspects of the system;
- The client has direct access to their data, making it easier to run analytics and generate reports. Given good system architecture there is also less vulnerability of losing data as a result of unforeseen circumstances to the vendor.

As already alluded to, Enterprise software often has a high up-front cost. However, its running costs should be lower than SaaS, albeit not zero since hardware needs to be maintained and there will inevitably be costs associated with supporting the software itself, either internally or paid to the vendor as part of a care package over the product's lifetime. With this in mind, overall cost of ownership of Enterprise software should converge with SaaS over time, even if this takes several years to achieve.

Some well known examples of this type of Enterprise software include Microsoft Dynamics and Microsoft Exchange.

More generally, Enterprise software is often associated with **Enterprise Resource Planning (ERP)** software, which provides solutions for accounting, billing, human resource management, corporate performance monitoring, project management, corporate governance, customer relations management and so on - effectively anything used to manage and analyse ongoing business activities. Sometimes this extends to ensuring that each particular piece of data is only stored in one data store, allowing multiple systems to rely on it as a definitive source of truth - doing so can improve reporting of business activity.

Bespoke Software

Another approach to cloud software provisioning is to write it yourself. So called **bespoke software** can be developed internally by a company or outsourced to a third-party. The latter is particularly common for organisations whose primary focus is not in IT.

Bespoke software offers a high level of flexibility because it can be built precisely to your requirements and not have to compromise with the needs of other users. Unfortunately, it can also be extremely expensive as you will have to take on responsibility for all parts of the software development lifecycle.

This means taking on the full costs of requirement analysis, system design, and the initial application development. This can take weeks of effort, even for relatively simple applications. Complicated programs or particularly high-quality UX can take even longer.

Once the bespoke application is deployed there will be a number of other costs. Internal training may be a big cost, especially if training material are required as there will be no prior art to make use of. Also, it is almost inevitable that additional features will be required after the initial release.

Even for a "feature complete" application, over time the infrastructure and underlying code will need maintenance, be it hardware replacement or patching software libraries and frameworks with security updates. This routine maintenance changes the software and its environment, often leading to "software rot" - i.e. the platform ceases to function properly because something about its environment or patched libraries do not work as expected. Further work to bring the application up-to-date is necessitated when this occurs.

With this in mind, opting for a bespoke option should be done with some caution. Nevertheless, with good planning it may be the best choice. A solution which exactly meets the client's need is extremely beneficial. It may also be the only way to support niche activities and processes which are specific to a particular organisation. Finally, in some cases a bespoke solution will be desirable in order for the client to retain control of business-critical intellectual property (IP) or because they are creating a novel product which they intend to resell.

Hybrid Offerings

So far in this section we have addressed SaaS, Enterprise and Bespoke solutions as clearly distinct approaches. This is useful for considering different types of cloud software, but in reality the dividing lines are quite blurred.

SaaS-Enterprise Hybrids

An obvious evolution of Enterprise solutions is for them to be hosted in the public cloud rather than on dedicated infrastructure. In some cases, vendors will also provide this as a service. There is software which is available both as SaaS and self-hosted.

Examples include:

- Atlassian offer their project management tools, Jira and Confluence in both self-hosted and SaaS forms;
- There is an overlap between the functionality of Microsoft Exchange and Microsoft 365, the former providing an Enterprise solution and the latter a SaaS product;
- Sage Cloud offers Enterprise-level ERP software but hosted in the public cloud.

Enterprise-Bespoke Hybrids

With many Enterprise vendors offering high levels of customisation, there is clearly a grey area between some Enterprise products and bespoke developments. Indeed, some "bespoke" solutions build directly on top of a pre-existing product.

There are a variety of systems which offer a core platform but which are designed to be extensible in this way. Examples include Sitecore and Umbraco (both extensible content management systems), and Microsoft Dynamics (extensible ERP software). Often, a wide range of vendor or third-party add-ons accompany this type of system and allow clients to customise them more easily. Alternatively the client can build their own plugins to deliver exactly the functionality they need.

Chapter 5

Cloud Certification Requirements

The major public cloud providers' offerings are all extremely wide ranging and complex. Consequently, it is perhaps no surprise that they all offer certification programs for their platforms.

Generally, certification programs have a number of levels ranging from basic qualifications indicating a high-level familiarity with the provider's service, through to very specific qualifications which demonstrate expertise in a particular area of focus. The latter may be achieved by following a multi-step pathway prescribed by the provider, with intermediate qualifications attained en-route. Assessment can be by means of hands-on practical evaluations and formal written / multi-choice examinations.

Clearly there are a variety of reasons why it is beneficial for individuals to complete this process:

- Increased confidence when using a particular provider's services;
- Qualifications demonstrate a level of expertise to employers;
- Organisations require a certain number of employees with certifications in order to qualify for partnership status (see below).

It should be noted that in all cases, certification expires after a period of time and must be refreshed, reflecting the rapid pace of change in cloud technologies.

Each large cloud provider has a partnership program. In these, organisations must meet a set of criteria which demonstrate to the cloud provider that they have a certain level of experience and expertise using the provider's services. Once these criteria are met, the cloud provider adds the organisation to a directory of partners which they make publicly available. Often partnership status is awarded in a particular area of focus, demonstrating that the organisation is expert in working with a particular aspect of the cloud ecosystem.

There are numerous advantages derived from partnership programs:

1. The directory of partners makes it easy for businesses to find organisations with the skills to develop cloud solutions;
2. Partner organisations gain a level of trustworthiness, which is reassuring for customers;
3. Cloud providers can funnel enquiries to their partners. This means the vendor does not need to maintain a consultancy business themselves, and the partner organisations gain more work;
4. The arrangement results in more users of the vendor's cloud ecosystem.

AWS

Amazon Web Services' certification process is based around "learning paths" which bring individuals to a recognisable level of expertise. Some paths are orientated around role, for example Developer, DevOps, or Architect. Others are focused on particular areas of expertise, such as Machine Learning, Networking, Game Tech and so on.

For more information, see AWS Certification.

Azure

Microsoft's certification process is similarly built around paths, although each path tends to have fewer stages than the equivalent AWS offerings. It should also be noted that Microsoft's certifications also include wider Microsoft services, in particular the 365 platform.

For more information, see Microsoft Certifications.

GCP

Google Cloud do not build their certifications around paths. They offer a set of qualifications in different subject areas. Completion of the exam and any supporting additional requirements such as practical assessments immediately confers the qualification.

For more information, see Google Cloud Certifications

Additional Material

Specialist tools

This section has mainly focused on more generic tools and concepts which are found in many different cloud architectures. The topics discussed form the backbone of cloud platforms which an engineer might create. These services are almost ubiquitous and can be applied to a wide range of applications.

Larger cloud providers additionally offer specialist services. These are beyond the scope of this module, but we highlight a few here for interest and to give a feel for the extraordinary breadth of tooling which is now available online on demand.

- Translation services: traditionally online translation services have used rule-based algorithms. However, more recently machine learning is being applied to the problem to create realistic results. Examples include Amazon Translate, Azure's Translator and Google Cloud Platform's Translation service (which powers Google's well-known online translator).
- Machine Learning (ML) and Artificial Intelligence (AI) services are becoming increasingly common. Offerings such as Amazon SageMaker provide user-configured services which can be applied to arbitrary tasks, acting as the "brain" of an overall data pipeline. Additionally, pre-trained AI Services are available off-the-shelf to deal with a range of more common scenarios. We have already mentioned translation services, but other pre-built AI solutions include text analysis (including sentiment and insight analysis), chatbots, trend analysis and image search to name a few.
- Blockchain-based applications have become increasingly popular in recent years. Although bitcoin is the most infamous blockchain, the underlying technology is applicable to any situation where an immutable record of a series of transactional events needs to be maintained. Cloud providers offer ledger databases for storing transaction chains, and managed infrastructure for creating blockchain networks.
- The **Internet of Things** (IoT) is the term used to describe the proliferation of network and internet-connected devices across the globe. It is an important growth market as more and more industrial and commercial users come to appreciate the benefits of

real-time telemetry, and as devices become widely installed in domestic environments. Cloud providers are coming to offer entire IoT ecosystems which allow data capture, virtual modelling of connected devices, IoT device operating systems, messaging, security, analytics and more.

Cloud Pitfalls

As with any technology choice, cloud computing has its pitfalls. The following list is not intended to be comprehensive, but highlights some common challenges associated with building an application in the cloud.

Serverless architectures present a number of challenges which need to be handled:

- Cold starts have historically been a common problem. These occur when code has not been run for a period of time and it is removed from the serverless system's memory. Afterwards, in order to run it again, the runtime environment has to be rebuilt, resulting in a substantially increased latency for the first request to the code. This can be particularly severe for languages such as Java which have comparatively heavyweight runtime environments. A naive solution to this is to keep code warm by periodically forcing it to run. Alternatively, serverless features such as AWS Lambda's Provisioned Concurrency can keep functions in a state of readiness, albeit at a monetary cost.
- Services such as Lambda generally have a maximum run time. If a code operation exceeds this time it will be prematurely terminated. For example, an HTTP request that relies on a database query or a call to an external service might cause a Lambda to time-out. This can be handled using an asynchronous HTTP polling pattern: the Lambda responds quickly to the initial request to indicate that processing has started, returning an identifier which the client can use to make subsequent polling requests to check the status of the processing job.
- Serverless systems are designed to scale rapidly and have many instances of the same application running in parallel. This necessitates a

different design of application from more traditional server-based solutions. In particular, applications should be stateless, and all operations should be designed in the assumption that there are a cluster of computers all working on the same dataset at the same time.

In general, the parallelisation of compute resources creates new challenges. For some applications, there are well established technologies for dealing with this. For example, map-reduce techniques exist in the data processing arena, whereby the data is broken up into suitable chunks allowing multiple systems to work on it at the same time.

Scaling in general needs to be undertaken with care. It is comparatively straight forward to scale compute resources. However, other areas of a system might create bottlenecks. A common constraint lies in SQL databases, which can be more challenging to scale. Wherever a bottleneck is encountered it will be necessary to employ suitable engineering approaches to deal with it. In databases, this may be employing read slaves, sharding data (splitting it between multiple databases based on some appropriate criteria) and so on.

All of this means that cloud computing should be regarded as a more complex engineering challenge than more traditional architectures. There are no insurmountable problems, but competent engineers are needed to set up a reliable, secure system.

PaaS Case Study: Heroku

Heroku is a good example of a cloud PaaS offering which simplifies the development process.

At its heart, Heroku aims to make it easy to deploy applications to the cloud. It achieves this by abstracting away the underlying infrastructure (which runs on AWS), so developers need only to deal directly with local configuration and deployment commands.

Applications can be developed locally in a variety of languages including Node.js, Ruby, Java, PHP, Python, Go, Scala and Clojure (all supported natively), plus others via third-party "buildpacks". Once written, a Heroku command line interface (CLI) tool is used to deploy the application into the Heroku ecosystem.

The main interaction with Heroku itself comes via the CLI. Applications can be deployed and configured, their status reviewed, user access managed and a comprehensive range of other tasks performed on the command line. The Heroku CLI is installed locally, and users must log in to the Heroku account associated with the application they are working on. Once this is done, interaction with the Heroku environment is seamless.

For example, seeing how many dynos (Heroku's name for a container) are running for a particular application is as simple as calling:

```
heroku ps
```

Heroku also offer an online dashboard interface. However, this is not really intended for development use - it allows account and user management, high-level application management, and reviewing application metrics, status and notifications. Use of the CLI is encouraged for day-to-day development activities.

Applications are deployed from the local development environment either using git to a Heroku remote, e.g:

```
git push heroku master
```

or through direct integration with GitHub or via the Heroku API.

Heroku's core offering is quite simple, concentrating principally on deploying code to one or more dyno (increasing the number of dynos obviously allows the deployed platform to scale). However, additional functionality can be added by means of an add-on system which provides access to a range of additional services including databases, logging, analytics, mailing and so on. Some important add-ons are made available by Heroku themselves, but many come from third-parties via an add-on marketplace, allowing the community to support itself in meeting particular needs.

Each time an application is deployed to Heroku, it is built into a **slug** which includes source code, dependencies, configuration environment variables, information about add-ons and so on. A slug is run on a dyno. Each version deployed is stored as a release, allowing easy rolling back to earlier versions should the need arise. Every running instance of an application uses the same slug, and shares add-ons (which are attached to applications) and environment variables.

There are clearly a number of advantages to using Heroku or a PaaS similar to it:

- It makes deployment of applications into a cloud environment extremely easy. Developers do not need to be intimately familiar with the cloud services themselves. Instead they need only to understand the principles which underlie Heroku and how to use its command line tools. "It just works".
- Deployment is fast and agile. Deployments can be torn down and replaced quickly, allowing for risk-free experimentation and iterative deployments.
- Scalable production systems are also easy to create. It becomes a case of setting scaling limits and deploying to suitably powerful dynos.
- Many "bread and butter" tools such as monitoring, logging, CI and security features are available off-the-shelf via Heroku, fully integrated into the deployed application. This removes a lot of the pain of setting these things up.
- All of the above arguably lead to reduced need for organisations to employ a DevOps specialist, although it should be noted that it's important still to have someone who understands how the application and deployment have been set up and deployed.
- Standard local development tools can be used to build applications. Development is done locally and then applications are deployed to Heroku. This can reduce vendor lock-in because the application is independent of the cloud architecture.
- Because the service is based on dynos, each dyno being a known size and costs, using Heroku can be a way to have predictable hosting expenses.

- Heroku themselves provide a high level of support to their customers, providing a mechanism by which problems can be quickly addressed.
- All in all, Heroku allows developers to focus on application development rather than operations.

However there are a number of disadvantages which offset the benefits:

- The simplicity of Heroku can also be a problem since it is highly opinionated about how cloud applications should be architected. Customisation is difficult. By abstracting the underlying infrastructure layer, the level of control over how it is configured is reduced which can become problematic.
- Related to the previous point, developers are to some extent limited to the functionality and tooling which is available off-the-shelf. While this is wide reaching, eventually a scenario will be encountered which Heroku cannot address. It is therefore probably not suited to highly complex Enterprise-level cloud solutions where there are specialist use cases, extremes of scaling or bespoke workflows.
- A common complaint levelled against Heroku is that it gets expensive once applications have a high volume of traffic, consume a lot of processing power, or if they need good resilience (high availability, low latency). There is a considerable price jump between basic production dynos and highly performant ones, with nothing in between.

Although Heroku does not publish details of the underlying infrastructure, it is widely understood that it is based on AWS. In essence then, they are reselling cloud services as a service, offering a really easy-to-use wrapper around a more complex and feature-rich platform. They add value by improving the developer experience - making operations very straightforward.

For more information about Heroku, see https://www.heroku.com/.