Yannis Viniotis, Ioannis Papapanagiotou

Cloud Architecture

ECE547/CSC547 Notes

Fall 2023

Second Edition

June 26, 2023

To Maria, Arabella, Banksy, Mila and Rhodes
(Yannis Viniotis)
To Sofia, Andreas and Vasiliki
(Ioannis Papapanagiotou)

Preface

This is a collection of notes used in the ECE547/CSC547, Cloud Computing course during the Fall 2022 semester. This material is complemented by handouts in the course website. Material that is copyrighted or that comes from an easily accessible source with stable content (e.g., Standards specifications, Wikipedia, AWS Support Center) is simply referenced in the text.

Sections marked as "This section is a Stub" will not be covered in the Fall 2022 semester. They have been included in order to complete the bigger picture. Let us know if you find material that fits into one of those stubs!

The narrative of this course

Here is, in a few paragraphs, what this course is about.

We start with the assumptions that you (the "audience", the student in the course): (a) are not an expert in the cloud computing field, (b) you are interested in working in it after graduation, and, (c) that you want to put a "strong" bullet in your resume.

So, *we assume that you have a few "silly questions*[1]*" in your mind.*

Question #1. What exactly is cloud computing?

Answer #1. That's the subject of Chapter 1. Assuming that assumption (a) holds true, a great way to address this question is to first have a crisp definition and examples of what computing means; what cloud means; and, finally, come up with examples of what is "non-cloud" computing.

Question #2. Who is hiring in the field of cloud computing? What are they looking for in a potential employee?

Answer #2. There is a diverse ecosystem that consists of five potential employers; we discuss in detail in Chapter 1. They are looking for a variety of skills. Their HR departments advertise for such skills using role names that are not consistent; so, if assumption (a) is true the roles are confusing. We'll clarify the roles in Chap-

[1] Another word for such questions is fundamental; so nothing to be ashamed of.

ter 1 and Chapter 6. In these notes, we'll address in some detail only two of these roles and this gives rise to the next two questions.

Question #3. How can I design a cloud computing system?

Answer #3. The direct answer to this question is given in Chapter 6. If assumption (c) is true, we need to lay some background, in order to fully appreciate the answer. First, we need to know what is a system in the cloud computing field, hence Chapter 2. Second, good design is tremendously facilitated by analysis of existing systems, hence Chapter 3. Third, **systematic** design starts with clearly stated goals, follows a **process** and applies tried-and-true **principles**; not so experienced architects can benefit from **best practices**. This third reason gives rise to Chapter 4.

Question #4. How can I automate operations in a cloud computing system?

Answer #4. The direct answer to this question is given in Chapter 5.

Figure 0.1 puts all these points together. It also captures the thought process in the authors' minds, that created this course.

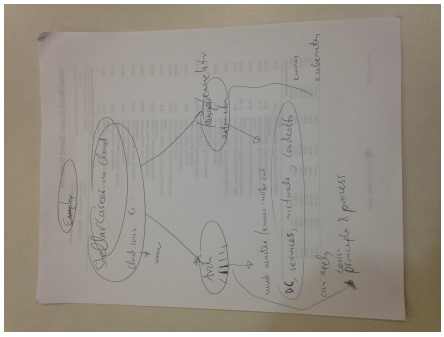


Fig. 0.1: The narrative in this course; rough sketch but you get the idea.

Note organization

These notes are organized as follows. In Chapter 1, we describe job roles and required skills in the cloud computing ecosystem. In Chapter 2, we present a brief introduction to (public) datacenters. In Chapter 3, we present the main models of cloud computing and discuss design problems that are specific to each. In Chapter 4, we propose a generic process for systematic architecture, some principles one can use in deriving such an architecture and best practices the (novice) architect can follow. We follow up on these generics with Chapter 6, in which we discuss some specific design problems, discuss the challenges and present possible solutions along with their tradeoffs. Finally, in Chapter 5, we present some topics of interest to cloud automation engineers.

Cary, NC *Yannis Viniotis*
Summer 2022

San Jose, CA *Ioannis Papapanagiotou*
Summer 2022

Acknowledgements

We would like to thank Snowflake (`www.snowflake.com`) for their inaugural Snowflake Faculty Fellowship award to Prof. Yannis Viniotis. This award has supported the preparation and writing of these notes.

The first edition of these notes was finalized in the serene environments of Linville, NC and Milos, Greece, during the summer of 2022. The second one was finalized in Milos, Greece, during the summer of 2023.

(a) Linville Falls, NC USA.

(b) Kleftiko, Milos, Greece.

Fig. 0.2: Where books should be written...

Contents

List of Figures

List of Tables

Topic 1
Job roles and skills in the field of cloud computing

Objectives: After reading this chapter, you should be able to: (a) define cloud computing, describe its key characteristics and distinguish it from similar concepts, (b) recognize the types of actors (i.e., potential employers) in the cloud computing ecosystem, and, (c) understand the differences between various technical roles you will be called upon to play in the ecosystem.

In this chapter, after we define what "cloud computing" is, we provide a brief overview of the (five types of) "players" in the cloud computing ecosystem: these will be potential employers after you graduate. They need/require a vast array of skills and will call you to assume a number of different technical roles. We present lists of roles and skills as they have appeared in the open, commercial literature. As you'll see, there is a fair amount of confusion in both; we supply our own descriptions in order to set up the background for the remaining topics of these notes. In the lectures and lab sessions of this course, we'll focus only on two roles, the (cloud) architect and (cloud) automation engineer respectively.

1.1 What is cloud computing?

Let's start by agreeing on what one can mean with the term "computing". After all, if there is cloud computing, there must also be plain vanilla, "non-cloud computing".

Loosely speaking, *computing is the use of resources to run (software) applications.* Examples of resources would be CPUs and main memory in servers, disk storage and the networks that connect users, CPUs and storage together.

Associated with a computing system are two entities: the one who is *using* it and the one who is "*managing* or administering" it. Here management involves a plethora of tasks like configuration, troubleshooting, upgrading, maintaining, etc.

Example 1.1. **Text processing on your laptop.** Using the MSword software package on your *personal* laptop is a very simple example of computing. The resources are the CPU in your laptop that runs MSword; the memory that houses the runtime package; the local disk that stores the software and all of your Word (.doc, .docx) documents. You are the *user as well as the administrator*. △

Several descriptions of the term "cloud computing" have been given, since its inception. We present some of them here and settle on one that is sufficient for our purposes.

1.1.1 An artistic view

A picture, they say, is worth a thousand words. So, in Figures 1.1 and 1.2 we have 3,000 words of an answer to the "What is cloud computing?" question.

Fig. 1.1: An artist's view of "computing in the cloud".

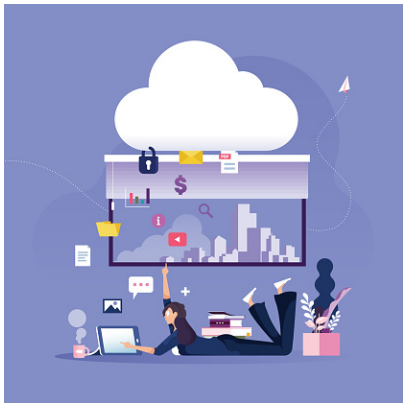
Hopefully, once you master the art of cloud computing, you can tell if any of these artists "got it right"...

1.1.2 The NIST (federal government) definition

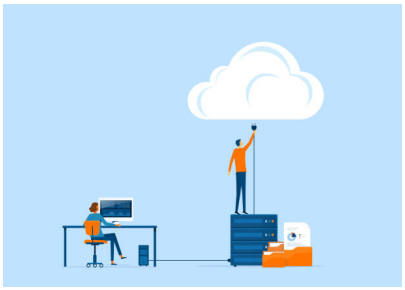
The National Institute of Standards and Technology (NIST) gave its definition in September 2011, a few years after cloud computing made its debut. The intended audience were system planners, program managers, technologists, and others adopting cloud computing as consumers or providers of cloud services.

We quote from [1] (the text in parentheses and emphasis is ours):

> **(The need for a definition.)** Cloud computing is an evolving paradigm. The NIST definition characterizes important aspects of cloud computing and is intended to **serve as a means for**



(a) "Worry-free computing", quite possibly by Rene Magritte.



(b) "Hey, I'm connected to the cloud!" Dadaism art movement, artist unknown.

Fig. 1.2: Other artists' views of cloud computing.

broad comparisons of cloud services and deployment strategies, and to **provide a baseline for discussion** from what is cloud computing to how to best use cloud computing.

(The definition.) *Cloud computing is a **model** for enabling **ubiquitous**, **convenient**, **on-demand network access** to a **shared** pool of **configurable computing resources** (e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned** and **released** with **minimal management effort** or **service provider interaction**.*

As we can see, this definition emphasizes the role of manager, the second entity involved in computing.

1.1.3 The Amazon and Microsoft (Industrial) definitions

Several years after the NIST definition, similar (but not identical) descriptions were given by major cloud computing providers. We mention two of them.

Amazon defines Cloud Computing in `https://aws.amazon.com/what-is-cloud-computing/` as

"*...the **on-demand** delivery of **IT resources** over the **Internet** with **pay-as-you-go pricing***".

Instead of buying, owning, and maintaining physical data centers and servers, you can access technology **services**, such as computing power, storage, and databases, on an as-needed basis from a cloud provider - for example, Amazon Web Services (AWS). ☺

As we can see, the Amazon definition emphasizes the cost associated with computing; in contrast to the NIST definition, it does not *directly* say anything about the management entity.

Microsoft defines Cloud Computing in `https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/` as

> *"...the delivery of **computing services**—including servers, storage, databases, networking, software, analytics, and intelligence—over the **Internet** ("the cloud") to offer faster innovation, flexible resources, and economies of scale.*

You typically pay only for cloud **services** you use, helping you lower your operating costs, run your infrastructure more efficiently, and scale as your business needs change.

The Microsoft definition emphasizes the cost associated with computing as well as some other characteristics; in contrast to the NIST definition, it does not say anything about the management entity, directly or indirectly.

1.1.4 Our (academic) definition

As you can see from the three definitions, there are similarities, differences, and, of course, terms that need further explanation; for example, does the term cloud in the Microsoft definition refer to the public Internet? or a private internet? Instead of dissecting the three, at this point in the game, we'll settle on our own definition (well, we'll borrow it from Wikipedia, actually) [5]:

> *Cloud computing is the **on-demand** availability of computer system resources, especially data storage (cloud storage) and computing power, **without direct active management by the user**.*

We chose this definition for its simplicity. It is good enough to get us started. The NIST definition is a "better" one; we'll revise ours as we gain more experience through the course.

Example 1.2. **Cloud or non-cloud computing? calculating π.** This number has captured imaginations of mathematicians and "common folk" since the ancient times. Knowing that π has an infinite number of digits satisfies the former; some

between the latter group of people want to know as many of those digits as possible[1]. According to [15], one such group that is 1,246 persons strong, used a collection of about 2,000 personal computers in 56 different countries to calculate that "...the least significant known bit of π is 1 at position 1,000,000,000,000,060 (one quadrillion and sixty).

Is this an example of cloud or non-cloud computing, in your opinion? What definition of cloud computing did you use? Justify your answer. △

Example 1.3. **Text processing in the cloud.** Continuing the discussion in Example 1.1, suppose that instead of installing MSWord on your laptop, you now use MSWord through Office 365, the cloud version. The software is not installed on your laptop; your .doc, .docx files are now stored in OneDrive, Microsoft's cloud storage service. You as the user do not worry about the vast majority of management tasks an IT person would have, related to storage (e.g., troubleshooting, backups, upgrades, configurations, etc.) And, if you think about it, you are not an expert in storage management, Microsoft personnel surely are better at it; that's an advantage for you. And, if you think more about it, Microsoft has superior buying power; they definitely buy disks for OneDrive at lower prices. Perhaps they pass on some of the savings to you. △

Example 1.4. **iphones and icloud storage.** Apple iphones have limited storage, due to the obvious phone size limitations. *icloud storage* reduces this limitation - for a fee, an iphone user can have access to much larger amounts of storage, somewhere in an Apple-managed datacenter, anytime they need it. The user does not worry about the vast majority of management tasks an IT person would have, related to storage (e.g., troubleshooting, backups, upgrades, configurations, etc.) △

1.1.5 A bit of history

Here is a silly question: what is the birth date of cloud computing? The Wikipedia reference [5] provides evidence that can support arguments of varying strength to place that date:

- somewhere in the 1960s, if one considers timesharing of IBM mainframes;
- 1994, with the offer of virtualized services;
- March 2006, with the offer of S3, the Simple Storage Service by Amazon.

Since Amazon popularized cloud computing and is a major player in the ecosystem at present, *we'll call March 2006 the birthday of cloud computing.*

[1] And some others hold world records for memorizing tens of thousands of them...

1.1.6 Characteristics of cloud computing

The Amazon and Microsoft definitions we presented mentioned (rather haphazardly) a number of (desirable) features of cloud computing. The Wikipedia article in [5] also mentioned even more in a quite confusing fashion[2].

NIST supplied a great list of features in [2]; we quote from page 2-1:

> Essential Characteristics:
>
> **On-demand self-service.** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
>
> **Broad network access.** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
>
> **Resource pooling.** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.
>
> **Rapid elasticity.** Capabilities can be rapidly and elastically provisioned, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
>
> **Measured Service.** Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

1.1.7 Similar concepts

We quote from the Wikipedia article in [5] (with some minor adaptations to improve readability):

> Cloud computing shares characteristics with:
>
> **Client–server computing.** Client–server computing refers broadly to any distributed application that distinguishes between service providers (servers) and service requestors (clients).
>
> **Grid computing.** A form of distributed and parallel computing, whereby a 'super and virtual computer' is composed of a cluster of networked, loosely coupled computers acting in concert to perform very large tasks.
>
> **Fog computing.** Distributed computing paradigm that provides data, compute, storage and application services closer to the client or near-user edge devices, such as network routers. Furthermore, fog computing handles data at the network level, on smart devices and on the end-user client-side (e.g. mobile devices), instead of sending data to a remote location for processing.

[2] Sometimes, you get your money's worth...

Utility computing. The "packaging of computing resources, such as computation and storage, as a metered service" similar to a traditional public utility, such as electricity.

Cloud sandbox. A live, isolated computer environment in which a program, code or file can run without affecting the application in which it runs.

Fog computing, in particular, has gained significant attention with the emergence of the Internet of Things (IoT).

1.2 Who are the potential employers in the field?

Who are the potential employers in the ecosystem of cloud computing? what skills do they need? what are the challenges for each one?

These are silly questions indeed, and of increasing silliness, we might say. We can fully answer only the first one in this chapter - the other two only partially.

1.2.1 The five actors in the cloud computing ecosystem

According to NIST, the National Institute of Standards and Technology, there are five entities involved in cloud computing (they are called organizational actors in [3]); they are depicted in Figure 1.3. Note that in these notes, we'll focus primarily on the center and top left actors (cloud provider and consumer).

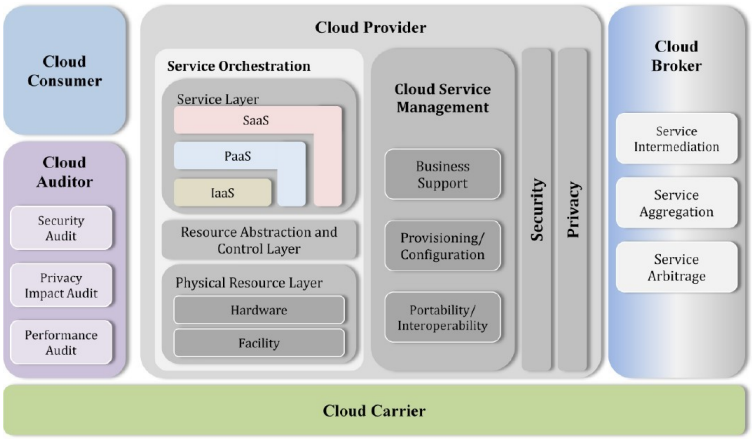


Fig. 1.3: The five actors in cloud computing [3].

We quote from [3] (the examples are ours):

The five actors, as defined in the reference model, are:

1.2.1.1 Cloud Consumer

Cloud Consumer is a person or organization that maintains a business relationship with, and uses service from, cloud providers. Anyone who uses (free or purchased) a cloud service is a consumer, and within the consumer there could be an array of roles responsible for configuring and managing the resources from the cloud provider depending on the services obtained.

Example 1.5. **Small-size fish: educational student accounts at AWS.** If you are not using one, as an NCSU student, do so now. Check `https://aws.amazon.com/education/awseducate/students/` and get started! △

Example 1.6. **Middle-size fish: NCSU as a consumer of gmail services.** A few years ago, faculty, staff and students at NCSU were using a variety of email systems. NCSU IT decided to switch all of us to gmail, saving, as their financial analysis showed, $2 per person, per semester. When you multiply that by about 40,000 consumers of email, it amounts to a significant chunk of $$$[3]. △

Example 1.7. **HUGE-size consumers, like Netflix.** Arguably, Netflix is the largest consumer of cloud services. Out of curiosity, read [13] for some Netflix stats. According to stat #21, AWS earns about $10M per month from Netflix. △

1.2.1.2 Cloud Provider

Cloud Provider is a person, organization, or entity responsible for making a service available to interested parties. A cloud provider would have a significant number of roles responsible for the management of its cloud resources including those responsible for selling, on-boarding, configuring and supporting cloud services for its consumers.

Example 1.8. **Oh no, another top 10 list...** Here is a list of the top 10 cloud service providers[4] according to [17].

The criteria used for the ranking does not matter for now; once we get to know the difference between IaaS, PaaS and SaaS in Chapter 3, we'll see that these are primarily IaaS providers. A top 10 list for PaaS or SaaS providers would be different.

1. Amazon Web Services (AWS)
2. Microsoft Azure
3. Google Cloud
4. Alibaba Cloud
5. IBM Cloud
6. Oracle
7. Salesforce

[3] At the time of writing of this example, the exchange rate was 1 USD = 74.69 INR = 0.88 Euros. So, by some calculations, NCSU saved millions per year...

[4] So you know where to send your resume, if you want to work for a cloud provider. Be advised though that there are several top 10 lists.

8. SAP
9. Rackspace Cloud
10. VMware

We'll supply some additional information about the top three in Section 3.4.1, page 60. △

Example 1.9. **Tool providers are significant providers too.** The description of the offerings of a cloud provider leaves out enterprises that supply "tools" to, say, providers and/or consumers. There are enterprises that supply middleware and management tools. Examples are: VMware (they provide VMotion, vRealize Cloud Management, etc.), Redhat (they provide Openstack, Opeshift, etc.). There are also enterprises that supply infrastructure for setting up a cloud. An example is Cisco (they provide switch fabrics, virtualization solutions like VxLAN and standards).

There is plenty of cloud related jobs in such organizations too. For our purposes, we'll place them along the cloud providers. △

1.2.1.3 Cloud Auditor

Cloud Auditor is a party that can conduct independent assessment of cloud services, information system operations, performance, and security of the cloud implementation. Generally, cloud auditors are categorized based on intent. For the most part, their focus is on risk and compliance, especially around information security. Other auditors can provide advisory services especially to consumers looking to cut down their bills or raise the level of efficiency in the resources consumed.

Example 1.10. **Security and cost auditors.** In the security category, we can mention Arctic Wolf Networks `https://arcticwolf.com/` and competitors; for cost auditing tools check [16]. △

1.2.1.4 Cloud Broker

Cloud Broker is any entity that manages the use, performance, and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers. Cloud brokers support consumers to get value for money by playing the advisory role especially for consumers who have a hybrid mix of resources from multiple providers.

Example 1.11. **Top 5 Best Cloud Brokers.** This is the last top list, we promise; in this chapter, at least. According to [12], the best five in 2021 were:

1. AWS Service Broker
2. IBM Multicloud Management Services
3. Cloudmore
4. Jamcracker Cloud Services Brokerage
5. Boomi

△

1.2.1.5 Cloud Carrier

Cloud Carrier is an intermediary that provides connectivity and transport of cloud services from Cloud Providers to Cloud Consumers. Most ISPs have taken the role of cloud carriers

as they provide the requisite bandwidth needed to connect consumers with providers as well as capabilities that support the connectivity.

Example 1.12. **AT&T carrying for Amazon.** In October 30, 2017, AT&T announced an agreement to be a carrier for AWS. AT&T offers "NetBond for Cloud" (check [14]) for carrying consumer traffic to providers' datacenters; AT&T and cloud providers jointly provision connectivity between the AT&T network and the providers' peering locations to access their public and private cloud solutions. △

1.2.2 The benefits and challenges for the providers and consumers of cloud computing

At this point, as Figure 1.4 implies, you should start having some questions pop up in your mind.

(a) What are the benefits and challenges for the **providers** of cloud computing?

(b) What are the benefits and challenges for the **consumers** of cloud computing?

Fig. 1.4: Questions, questions: where are the answers?

If you are as good an engineer/computer scientist as the one in Figure 1.5, you should definitely start pondering such questions. In Chapters 6 and 5, we'll study such questions and provide some answers. You can try your own, in the meantime.

Fig. 1.5: Did you know that Mr. Bean is an Electrical Engineer?

1.3 What are the technical roles (hats) in the field?

There are several technical job roles within each actor organization - *several "hats" you may wear in your career.* There is no consensus on titles for these roles as well as pertinent skills. This lack creates some degree of confusion, as we'll shortly see.

1.3.1 A random, (confusing) description of the technical roles

In this section, we follow the descriptions provided in [4]. In the problem section, we ask you to compare them to two different sources.

Note that Amazon, a leading cloud provider, refines the Cloud Architect role further; we'll discuss that refinement later, in Section 6.1, after we introduce their AWS Well-Architected Framework.

These are the seven technical roles in a cloud ecosystem partner, according to [4].

1.3.1.1 Cloud administrator

Cloud administrator: To become a cloud administrator, a candidate generally must have three to five years of practical experience in the cloud. A cloud admin should have a strong understanding of system management, troubleshooting and virtualization. They should know Linux, along with some configuration management tools, monitoring tools and scripting languages. Beyond their cloud knowledge, administrators should have strong leadership and people skills.

1.3.1.2 Cloud architect

Cloud architect: A cloud architect focuses on the big picture of infrastructure design and configuration rather than individual server configurations. To succeed, the candidate should bring eight to 10 years of experience and be able to build a roadmap for the organization's existing and future cloud assets.

New technologies can affect the company's cloud infrastructure. A cloud architect must have the foresight to see how those changes and emerging technologies will affect their systems. It helps for a cloud architect to have a few certifications under their belt, whether completed through independent training or appropriate companies.

1.3.1.3 Cloud engineer

Cloud engineer: A cloud engineer is in charge of any and all technical responsibilities associated with cloud computing. Companies that want to hire a cloud engineer look for someone with three to five years of cloud services experience. The engineer should be versed in areas such as open source technology, scripting languages, multi-cloud environments, system engineering and software development. This cloud computing job role is responsible for the design, planning and management of the cloud infrastructure. Familiarity with APIs, orchestration and automation, DevOps and databases are all pluses for cloud engineers on top of their computer science or engineering degrees.

1.3.1.4 Cloud security manager

Cloud security manager: Security is a concern for both private and public clouds. Providers and users alike take comprehensive security measures to ensure that data is stored safely. Because of this, cloud security roles are vital for IT teams within companies. Cloud security managers should have completed formal training and acquired vendor-neutral certifications to stand out in the job field.

To be competitive, a cloud security manager candidate should have a strong understanding of compliance issues and IT governance related to the cloud. A security manager designs, implements and maintains security strategies. They should know the major modern software development approaches, because the role is often incorporated into software development. Security managers can excel in the position if they constantly monitor the IT landscape to assess and prevent new threats to the corporate cloud estate.

1.3.1.5 Cloud application developer

Cloud application developer: An effective cloud application developer has to be a proficient in most – if not all – major scripting languages, with typically at least five years of experience. However, specific job roles dictate the expected software tools knowledge. For example, if the job is geared toward web development, demonstrate skills with HTML5 and jQuery. Since software development requires integration tasks, application developers need to know the back-end system integrations with the major cloud platforms.

Cloud app developer applicants should research the common cloud providers and platforms. Someone in this role will build, test and deploy applications in a company's cloud environment – often using DevOps practices, as well as CI/CD tools. If candidates are experienced, versatile and work well with others, they are a great fit for a cloud application developer position.

1.3.1.6 Cloud network engineer

Cloud network engineer: A cloud network engineer wears many hats. The role is primarily responsible for the implementation, configuration, maintenance and support of the entire cloud network – but the obligations don't stop there. Cloud network engineers can also be in charge of the administration, monitoring, documentation, security and integration of the company's network, as well as other related cloud services.

A good network engineer has acquired relevant certifications in networking, security and other industry standards for the job role. Candidates should have practical experience in asset deployment and management. This role requires a strong understanding of data center administration – preferably more than five years of experience.

1.3.1.7 Cloud automation engineer

Cloud automation engineer: One of the most critical and influential roles that affects a business's success with its cloud strategy is a cloud automation engineer. An automation engineer takes experience from software development or IT operations positions and applies a focus on cloud automation, orchestration and integration. More often than not, this role requires a widespread understanding of hardware and software, as well as data center and cloud infrastructure. A cloud automation engineer implements, optimizes and supports an infrastructure.

To get hired in this growing cloud computing job role, a candidate should have five to 10 years of experience in infrastructure operations and application development, in addition to two or more years of practical experience with CI/CD development models. There are no industry certifications required for this position across the board; expectations vary from employer to employer. A good cloud automation engineer has hands-on experience with cloud platforms and technologies. This is usually a senior position, so it is crucial for the engineer's knowledge and experience to be reflected in her work.

1.3.2 Our own description of the technical roles

A closer examination of the responsibilities of each role, as described in the previous section, creates a bit of confusion. For example, how is the admin different from

the network engineer role? Does the cloud security manager do both design and implementation? If only implementation, how does that differ from the admin role? is there an intersection between the responsibilities of an automation engineer and a network engineer, since there is automation in networks as well?

In order to avoid confusion when we discuss the topics in Chapters 6 and 5, we provide next our own taxonomy of the technical roles and mention explicitly their association with an actor in the ecosystem. As expected, the skills required depend on the actor. We do not specify these skills in detail, since we'll only focus only on two roles in these notes.

1. **Administrator.** This "hat" works for a consumer, provider, or carrier.
2. **Architect.** This "hat" works for any one of the five actors in the ecosystem. The typical example is the architect who works for a cloud provider like Amazon; or a cloud consumer like the fishes in Examples 1.6 and 1.7. Another example would be an architect who works for an enterprise that supplies middleware and management tools - such as, VMware (they provide VMotion, vRealize Cloud Management, etc.), Redhat (they provide Openstack, Opeshift, etc.).
3. **Automation Engineer.** This "hat" works for a consumer, provider, or carrier.
4. **Application Programmer.** This "hat" works for any one of the five actors in the ecosystem. Note that applications can run in the user, control or management plane.

We will focus our lectures on hats #2 and #3. In the labs we will see some of the pains hat #3 endures.

1.4 The cloud architect hat

This is the first hat we'll focus on in this course.

1.4.1 What does a cloud architect do?

Let's see first what this hat does for a living, in "industry-speak" words (Figure 1.6 summarizes succinctly).

The authors of [6] were a bit more loquacious[5]. We quote them (the text in blue is ours):

> **WHAT DOES A CLOUD ARCHITECT DO?** *A Cloud Architect is responsible for converting the technical requirements of a project into the architecture and design that will guide the final product. Often, Cloud Architects are also responsible for bridging the gaps between complex business problems and solutions in the cloud. Other members of a technology team, including DevOps engineers and developers, work with the Cloud Architect to*

[5] *loquacious?* full of excessive talk; wordy; given to fluent or excessive talk; in plain English, ***garrulous***.

Fig. 1.6: What does the cloud architect do?

ensure that the right technology or technologies are being built.

Key Traits of Cloud Architects - None is technical

- Strategic: Cloud Architects need to be able to identify fit-for-purpose technologies.
- Planner: As project manager, the Cloud Architect must be able to guide the development of a solution from beginning to end, accounting for any roadblocks and complexities.
- Clarity: Ability to understand and communicate elegant solutions to complex problems. Cloud Architects must be able to identify patterns.
- Detail-Oriented: With so many moving parts in the development process, a Cloud Architect must be able to both understand and communicate the nuances of business requirements, functionality, and maintenance requirements.
- Collaborative: As the center of the development process, a Cloud Architect must be able to lead and influence stakeholders within the organization.

Day-to-Day Expectations and Responsibilities - What skill set is needed to do a good job here?

- Automate infrastructure and build pipelines for continuous integration and continuous deployment
- Work with business stakeholders to translate requirements into application architecture
- Stay abreast of industry trends, emerging technologies, and software development best practices
- Develop cloud adoption plans, application design, and deployment mechanisms

Cloud Architect Target Technical Skill Set

- Deep knowledge of cloud platforms and services
- Deep understanding of software design patterns
- UNIX and Linux experience
- Knowledge of DevOps tools and methodologies
- Knowledge of emerging and existing technologies

Cloud Architect Target Non-Technical Skill Set - Shouldn't this be under Key Traits?

- Ability to communicate risk, reward, and complex concepts to diverse stakeholders
- Ability to persuade and contextualize
- Ability to think strategically, develop plans, and coordinate execution

1.4.2 In our words, what does a cloud architect do?

In Section 4.4.1, we provide a generic, summary view of what a (generic, not just cloud) architect does, after we define formally what business requirements are; they depend on the actor organization. For now, Figure 1.7 summarizes, once again, succinctly.

Fig. 1.7: What does the cloud architect do? In our own words...

We explain in some detail next.

1.4.2.1 Architect for a (public) cloud provider

In a nutshell, this architect[6]:

- "puts together" the infrastructure inside a datacenter. For example,
  1. chooses hard elements like servers, storage, switch fabrics;
  2. chooses soft elements like operating systems, middleware, management software, protocols;
  3. sets operational policies;
  4. chooses power plant, cooling;
- "puts together" the network of datacenters. For example,
  1. selects the number and locations of the datacenters;
  2. designs the interconnection of these datacenters;
  3. designs the network that connects tenants to their applications inside the datacenter;
- "puts together" the service portfolio. For example,
  1. defines the services exposed to tenants;
  2. sets up the SLAs (and pricing?) for such services;

[6] In Section 6.1.1, we'll add some more responsibilities to this role; for now, what we mention in this section will suffice.

3. designs how datacenter resources are shared among tenants;
4. designs how datacenter resources are protected from misbehaving tenants;

1.4.2.2 Architect for a cloud consumer

In a nutshell, this architect:

1. selects services offered by a cloud provider, in order to meet business objectives. Such top-level objectives include:
   - cost;
   - security and compliance concerns;
   - performance;
2. designs how acquired datacenter resources are shared among the enterprise applications;
3. designs the (private) network that connects applications to their users;

1.4.2.3 Architect for an enterprise that supplies middleware and management tools

In a nutshell, this architect:

1. designs middleware tools, such as:
   - Terraform;
   - Prometheus;
   - VMware Motion;
2. designs management tools, to be used, for example, in monitoring, cost analysis, configuration.

1.4.3 What skills does a "good" cloud architect have?

Hmmm...Define "good". That's a million dollar question. We'll start with the (unaltered) view expressed in [7].

Author: Dror Helper

> Cloud architects are projected to be the second most in-demand tech job in 2021. This role is responsible for designing and developing advanced cloud-based solutions for organizations migrating their existing workloads and infrastructure to the AWS cloud. Using AWS, cloud architects have limitless virtual resources, which can be quickly provisioned and disposed. It can be overwhelming; there are many services you need to become familiar with and, on top of that, having infrastructure and data in the cloud can become a security

nightmare if not handled correctly. Here are seven skills cloud architects need to rock this in-demand role.

1. Java, Python or C# Most architects have a software development background. An efficient AWS architect should be able to write code in Java , Python , C# or any other of the programming languages which have an official AWS SDK. Understanding programming in general is important for creating viable, logical solutions that would work as intended. And a good architect can use programming to quickly create a proof of concept or demo to show a point or investigate how to use the latest and greatest technologies.

2. Networking It's hard to create a secure, scalable cloud-based solution without understanding networking. DNS, TCP/IP, HTTP, CDN and VPN are only a few of the terms you want to make yourself familiar with. That doesn't mean you need to know the port that you need to open for SSH access (although it helps). As an architect you're expected to be able to use services such as Route 53 (DNS), CloudFront (CDN) and Virtual Private Cloud (VPC) to design your cloud networking using public and private subnets, internet access and VPC peering.

3. Data storage fundamentals Every software architect needs to know and understand how and when to use databases. In AWS, where you have many data storage options available, you need to be able to know when to use each. From simple, yet powerful, bucket storage using S3 to Relational Database Service (RDS) and all the way to full fledged Hadoop clusters , you'll need to compare different capabilities, performance and price, and choose the best way to store some or all of your company's data.

4. Security foundations From securing access to your AWS account to securing access to your data, AWS has several services and guidelines created specifically to help you make sure only authorized code and people are allowed to perform specific tasks. You will also need to learn about Identity and Access Management (IAM), a service that will help you define which services and users can access which resources. Learn how to secure your networks using Security Groups and Access Control Lists.

5. AWS service selection Cloud architecture involves front-end and back-end technologies backed by components provided by a cloud vendor, in many cases Amazon. Good cloud architects should know what services are at their disposal and have a good understanding of the services relevant to the organization. This is no easy task since Amazon tends to release new services throughout the year. Knowing what to focus on is crucial. Basic services every AWS architect should know include SQS (simple queuing), SNS (notifications) and RDS (Relational Database Service). Knowledge of more specific ones—such as one of the AWS IoT related services—is extra helpful.

6. Cloud-specific patterns and technologies Once you move code into the cloud some rules change. Scalability, availability and recovery become easy—as long as you design your workloads and harness AWS infrastructure correctly.

Using messages, storing state in the right place and handling failures correctly are a big part of creating scalable and cost-effective applications. Patterns such as pub/sub, queuing and eventual consistency will help you create applications that can be scaled by creating more instances of the same service.

On top of that you'll need to choose between single applications to services or go serverless using AWS Lambda—all of which could provide cost-effective and performant solutions, depending on your needs.

7. Communication Designing great architecture doesn't mean a thing if you fail to explain your vision to software developers, managers and fellow architects. You'll need to learn to explain your ideas via emails, documents and presentations in a way that convinces your team why your solution is the best. Work on your presentation skills, learn how to write in a clear and concise way and use a diagramming tool to show complex environments. A single slide with a diagram is preferable to ten slides with twenty bullet points each.

In Problem 1.8, we ask you to investigate another view, that lists even more (10+8) skills.

1.4.4 What cloud architect skills will we address in this course?

Well, as you may have guessed, the technical skills are the content of Chapters 2 through 6. The Communication soft skills we'll address in homeworks and the class project.

1.5 The cloud automation engineer hat

This is the second hat we'll focus on in this course.

1.5.1 What does a cloud automation engineer do?

We quote from the blog in [8] a super-detailed list:

> The role that a Cloud Automation Engineer has to play in any organization is as follows:
>
> 1. Participate in the design of service automation in cloud towards Infrastructure-as-code.
> 2. Central to obtaining and processing requirements in order to transform application delivery for cloud in an agile fashion
> 3. Leader of orchestration in cloud using Infrastructure-as-code while providing self-service capabilities to IT teams
> 4. Active driver for Continuous Integration and Continuous Delivery applications
> 5. Maintain adherence to architectural standards/principles, global product-specific guidelines, usability design standards, etc.
> 6. Provide support and documentation to assist in sustaining projects during the transition to production.
> 7. Input into internal and external organizations towards development of standards as well as efficiencies
> 8. Provide feedback regarding development and engineering methodologies, standards and leading practices.
> 9. Involve and drive firm's evolution towards DevSecOps and Agile Transformation
> 10. Execution of process engineering and operational improvement initiatives for automation tooling focused on cloud
> 11. Resolve and act as escalation and coordination point for incidents and problems related to affected automation
> 12. Requirements gathering for design, development and deployment
> 13. Root cause analysis and improvement solutions
> 14. Executing scheduled or unscheduled automation in support of other technology domains
> 15. Selects appropriately from applicable standards, methods, tools and applications and use accordingly.
> 16. Ability to work well within a multi-disciplinary team structure, but also independently
> 17. Ability to work with 3rd party vendors (i.e., Microsoft, Amazon, etc.) for escalation of issues
> 18. Demonstrates analytical and systematic approach to problem solving.

1.5.2 What cloud automation engineer skills will we address in this course?

As you can probably see, the above list is quite detailed and confusing: it mixes architecture (e.g., requirements gathering) and "engineering" responsibilities. It specifies technical as well as soft skills. We will focus primarily on automation tools, as we detail in Chapter 5.

In a nutshell, our focus will be on the three operations around (a) cloud automation, (b) orchestration, and, (c) integration.

1.6 Problem Section

Problems on cloud computing definition and characteristics

Problem 1.1. Cloud definitions. Consider the NIST, Amazon and Microsoft definitions of cloud computing mentioned in Section 1.1, page 1.

1. The Amazon and Microsoft definitions mention the Internet as the delivery medium. Is this necessary?
2. The NIST definition mentions *a shared pool* of resources. Provide examples of entities who can share. Do you see any tradeoffs in sharing?
3. The NIST definition mentions *minimal* management effort. Which entity sees its effort reduced? Since "there is no free lunch in engineering", is there another entity whose management effort increases?

Problem 1.2. Cloud characteristics. Read the Characteristics section of reference [5]; it describes ten characteristics as opposed to the five ones mentioned in Section 1.1.6, page 6.

1. Which characteristics are common in both?
2. Which set of characteristics would you agree with? Justify your choice.

Problem 1.3. Cloud characteristics. Consider Example 1.2, page 4.

1. Is this an example of cloud computing? If yes, according to which definition?

Problems on actors, job roles and skills

Problem 1.4. Actors in the ecosystem. Give an example (different from the one we presented in the notes or discussed in class) of:

1. A cloud consumer.
2. A cloud provider.
3. A cloud carrier - google the term "Sprint cloud services", for example.

Problem 1.5. Technical roles. Read reference [3]; it describes only five technical roles. Map those roles onto the seven roles we discussed in Section 1.3. Justify your classification.

Problem 1.6. Technical roles. Read reference [9]; it describes seven technical roles. Map those roles onto the seven roles we discussed in Section 1.3. Justify your classification.

Problem 1.7. Technical responsibilities. Read references [3], [4] and [9]; they describe several responsibilities associated with technical roles.

1. List the union of these responsibilities.
2. Associate them with the seven roles we discussed in Section 1.3.

3. For each role, describe one responsibility in more detail.

Problem 1.8. Cloud architect skills. Read reference [9]. It describes "Top 10 Must-Have Skills for Successful Cloud Computing Career in 2021" as well as eight additional technical plus "soft" skills.

1. Identify the skills that are pertinent for the architect role.
2. In your opinion, which skill is the most important? Why?
3. In your opinion, which skill is the least important? Why?

Problem 1.9. Cloud engineer skills. Read reference [9]. It describes "Top 10 Must-Have Skills for Successful Cloud Computing Career in 2021" as well as eight additional technical plus "soft" skills.

1. Identify the skills that are pertinent for the Cloud engineer role. State clearly what is your definition of this role.
2. In your opinion, which skill is the most important? Why?
3. In your opinion, which skill is the least important? Why?

Problem 1.10. Cloud engineer skills. Read reference [10]. It describes "...what it takes to become a cloud automation engineer." Compared to [8], it proposes a much shorter list of responsibilities.

1. Reconcile the two lists. Are there responsibilities in [10] that are not mentioned in [8]?
2. (The importance of being accurate in descriptions - or being careful and critical of what you read.) Reference [10] mentions an "infrastructure stack" that cloud automation engineers must be very familiar with. In your opinion, how can the bottom layer of the stack ("Data center") be different from the three layers above it?

Problems on industry certifications and jobs

Problem 1.11. Cloud Computing Certifications, Admin role. Read reference [9]. It mentions several industry certifications that are available in the cloud computing space (the analogs of Cisco Certifications in the networking space). In this problem, we'll narrow the list to the "big three" cloud providers. (More on these providers in Section 3.4.1, page 60.)

Consider the Cloud administrator role.

1. Find information on the AWS certification exams. Supply a summary.
2. Repeat for Google Cloud.
3. Repeat for Microsoft Azure.

Problem 1.12. Cloud Computing Certifications, Architect role. Read reference [9]. It mentions several industry certifications that are available in the cloud computing space (the analogs of Cisco Certifications in the networking space). In this problem, we'll narrow the list to the "big three" cloud providers. (More on these providers in Section 3.4.1, page 60.)

Consider the Cloud architect role.

1. Find information on the AWS certification exams. Supply a summary.
2. Repeat for Google Cloud.
3. Repeat for Microsoft Azure.

Problem 1.13. Cloud Computing Certifications, Architect role. Read reference [11]; it provides information about the "beginner" AWS Certified Solutions Architect – Associate certification.

AWS recommends "...one or more years of hands-on experience designing available, cost-efficient, fault-tolerant, and scalable distributed systems on AWS." Ideally, after taking this course and some "summer practice", you should have enough hands-on experience and theoretical foundations to ace such an exam. ☺☺

1. Download the exam guide. Summarize what topics you will be tested on.
2. The Appendix of this guide lists several AWS services that might be covered on the exam. (More on these services in Section 3.4.2, page 63.)
   Find out and summarize the objective of the Amazon CloudWatch service.
3. Read topics 2.1 - 2.4 in "Domain 2: Design High-Performing Architectures". Select one topic and describe it in as much detail as possible.
4. Download the sample questions. Consider the first question on Load Balancing. Answer it without looking at the answer. Justify your answer.

Problem 1.14. Job advertisements. Search for job advertisements. Create a list of technical as well as non-technical skills you find in these advertisements. You can use your own source in addition to the following two.

1. Cloud providers; an example is Google Cloud Platform: `https://careers.google.com/cloud/`
2. All actors in the ecosystem: `https://www.indeed.com/jobs`

Huh?

Problem 1.15. Cloudy themes. Search the web for cloud-related but non-technical themes; you may want to google "cloudy themes" or "cloudy designs".

1. Submit a piece that you find humorous.
2. Submit an *oeuvre* by a painter. Explain what attracted your attention.

References

In every chapter of these notes, references marked as **Handout H1.x** are required reading material; *you are responsible for them come exam time*. You will find them in the required handouts binder in moodle. The rest are used in some form in the text or problems. You are encouraged to read them; your performance in the exam does not directly depend on them.

1. **Handout H1.1:** The NIST Definition of Cloud Computing, Special Publication 800-145, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
2. **Handout H1.2:** Cloud Computing Synopsis and Recommendations, Special Publication 800-146, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-146.pdf
3. **Handout H1.3:** Common Roles in Cloud Computing, https://www.bmc.com/blogs/cloud-computing-roles/
4. **Handout H1.4:** Sara Grier, 7 cloud computing job roles to advance your career, https://searchcloudcomputing.techtarget.com/feature/7-cloud-computing-job-roles-to-advance-your-career
5. **Reference R1.1:** Wikipedia article on Cloud computing, https://en.wikipedia.org/wiki/Cloud_computing
6. **Reference R1.2:** What Does a Cloud Architect Do? https://cloudacademy.com/cloud-roster/cloud-architect/
7. **Reference R1.3:** Dror Helper, AWS CLOUD ARCHITECT ESSENTIAL SKILLS, https://www.allhandsontech.com/cloud/aws/aws-cloud-architect-essential-skills/
8. **Reference R1.4:** Detailed blog entry on the responsibilities of a cloud automation engineer, Nov 15, 2018, by Priyaj https://www.edureka.co/community/30570/what-is-the-job-role-for-a-cloud-automation-engineer
9. **Reference R1.5:** Prajakta Patil, Chiradeep BasuMallick, "Cloud Computing Careers: Job Roles and Key Skills Needed for 2021", https://www.toolbox.com/tech/cloud/articles/cloud-computing-careers-job-roles/
10. **Reference R1.6:** Stephen J. Bigelow, "Here's what it takes to become a cloud automation engineer", https://searchcloudcomputing.techtarget.com/tip/Heres-what-it-takes-to-become-a-cloud-automation-engineer
11. **Reference R1.7:** AWS Certified Solutions Architect – Associate certification, https://aws.amazon.com/certification/certified-solutions-architect-associate/
12. **Reference R1.8:** Best cloud brokers 2021, https://www.itproportal.com/guides/best-cloud-brokers/
13. **Reference R1.9:** 50+ Netflix statistics & facts, https://www.comparitech.com/blog/vpn-privacy/netflix-statistics-facts-figures/
14. **Reference R1.10:** NetBond for Cloud services, https://www.business.att.com/products/netbond.html
15. **Reference R1.11:** PiHex, https://en.wikipedia.org/wiki/PiHex
16. **Reference R1.12:** Top 8 Cloud Cost Management Tools, https://harness.io/blog/cloud-cost-management-tools/
17. **Reference R1.13:** Top 10 Cloud Service Providers In 2021, www.c-sharpcorner.com/article/top-10-cloud-service-providers/

Topic 2
Datacenter Fundamentals

Objectives: After reading this chapter, you should be able to: (a) describe in some detail the components of a datacenter, (b) describe the notion of datacenter workloads and their competition for resources, and, (c) list some issues that must be addressed in resolving this competition.

Datacenters are the places where cloud computing takes place; by some estimates, this was around 8,000 places in the world, in 2021[1]. In this chapter, we provide a brief overview of all the components of a datacenter. We then narrow our discussion on only the components of interest to the two hats, in order to provide the necessary background for the next chapters.

2.1 Datacenter Basics

Loosely speaking, *a data center is space that houses IT resources*. Such resources include computing engines (e.g., CPUs, GPUs, typically packed as a server), storage (e.g., disks, tapes) and interconnection equipment (e.g., cabling, switches, routers, gateways). This space can be as small as a special room (e.g., a few hundred of square feet) in a building or a mega-facility such as the one shown in Figure 3.2, page 59. Datacenters over 10 million square feet have also been reported [3].

The site `https://datacenterlocations.com/` provides location information on a large number of datacenters. Google provides a map of their own datacenters in

[1] You might think that these places are on the surface of the earth. Well, at least one place is at the (not so deep) bottom of the ocean. Microsoft is experimenting with it `https://natick.research.microsoft.com/`. Word on the street has it that Elon Musk, in revenge to Jeff Bezos, has serious plans to experiment with a datacenter on Mars. So far, Elon has not replied to the authors' emails regarding these rumors.

https://www.google.com/about/datacenters/locations/. Reference [1] has some interesting information about datacenters worldwide; in particular, check the information on the size of data.

2.1.1 A bit of history

We quote from the Wikipedia article in [2]:

> Data centers have their roots in the huge computer rooms of the 1940s, typified by ENIAC, one of the earliest examples of a data center. Early computer systems, complex to operate and maintain, required a special environment in which to operate. Many cables were necessary to connect all the components, and methods to accommodate and organize these were devised such as standard racks to mount equipment, raised floors, and cable trays (installed overhead or under the elevated floor). A single mainframe required a great deal of power and had to be cooled to avoid overheating. Security became important – computers were expensive, and were often used for military purposes. Basic design-guidelines for controlling access to the computer room were therefore devised.
>
> During the boom of the microcomputer industry, and especially during the 1980s, users started to deploy computers everywhere, in many cases with little or no care about operating requirements. However, as information technology (IT) operations started to grow in complexity, organizations grew aware of the need to control IT resources. The availability of inexpensive networking equipment, coupled with new standards for the network structured cabling, made it possible to use a hierarchical design that put the servers in a specific room inside the company. The use of the term ”data center”, as applied to specially designed computer rooms, started to gain popular recognition about this time.
>
> The boom of data centers came during the dot-com bubble of 1997–2000. Companies needed fast Internet connectivity and non-stop operation to deploy systems and to establish a presence on the Internet. Installing such equipment was not viable for many smaller companies. Many companies started building very large facilities, called Internet data centers (IDCs), which provide enhanced capabilities, such as crossover backup: ”If a Bell Atlantic line is cut, we can transfer them to ... to minimize the time of outage.”[9]
>
> The term cloud data centers (CDCs) has been used. Data centers typically cost a lot to build and to maintain. Increasingly, the division of these terms has almost disappeared and they are being integrated into the term ”data center”.

2.1.2 Business and technology drivers for datacenters

In our opinion[2], the two main **business** drivers for the shift of IT infrastructure towards datacenters are:

1. Cost reduction (AWS has superior buying power; others)
2. The need to control the complexity of large pools of IT resources (AWS has superior technical expertise; others)

The two main **technology** drivers for the same shift are:

[2] We can spend valuable lecture time debating this, of course.

1. Server virtualization (enables consolidation, collocation of customers/tenants)
2. Fast networks enable connectivity of users and cloud applications

We'll see in the next section how similar drivers created the shift from physical infrastructure to a virtualized one.

2.2 The components of a datacenter

The components of a datacenter can be divided into four categories:

- *Physical plant*. This category contains the power plant and cooling. Both components contribute heavily into the operational expenses of running a datacenter. The sustainability requirement that Amazon introduced recently into their cloud requirements (more on those requirements later, in Section 4.3) relates heavily with the physical plant. We will not deal with this component in these notes[3].
- *Hardware*. This component involves compute and storage servers as well as the network that connects them together. Some (but not all) of the hats of interest to us deal with this component.
- *Software*. This component includes applications that the consumers run in the datacenter as well as control and management software that the providers run to keep everything humming. Like hardware, some (but not all) of the hats of interest to us deal with this component.
- *Human personnel*. This is all the people involved in running the datacenter. If the idea of a "lights-out" data center[4] becomes reality, this component will be reduced to perhaps only the tour operators. We will not deal with this component in these notes.

2.2.1 Hardware components

These components are also known as the physical infrastructure. Of interest to us are only two of them, namely the physical *compute* and *storage servers*. These servers are housing the consumers' software applications as well as the provider's control and management software.

[3] It is of interest to only one architect subspecialty, as we describe in more detail in Section 6.1.

[4] See `https://www.datacenterdynamics.com/en/analysis/what-lights-out-data-center/` for an interesting discussion; also available as reference [16].

2.2.1.1 Physical compute servers.

A physical compute server houses the VMs and containers we'll introduce in Section 3.1. We will discuss in the next chapters how providers expose such servers to their consumers. Unlike a server at home, a compute server in a datacenter does not typically have nonvolatile storage (e.g., a hard disk). In a nutshell, for the hats of our interest, a *physical compute server provides consumers with two resources: CPU and memory*.

Let's see some random examples next, in order to get an idea about the capabilities of some commercially available compute servers. It is a great exercise to find out the "most powerful" server a manufacturer has produced...

Example 2.1. **HPE ProLiant DL360 Gen10 Server.** For details, see the datasheet in `https://www.router-switch.com/media/upload/product-pdf/hpe-dl360-gen10-servers-d` `pdf`. This server supports the Intel Xeon Scalable Processor Family with up to 28 cores, plus a DDR4 SmartMemory supporting up to 3.0 TB max. △

Example 2.2. **Dell EMC PowerEdge Servers.** See `https://www.delltechnologies.com/en-us/servers/index.htm`. Once in there, you can play with the number of cores and memory size. △

Example 2.3. **Cisco UCS B200 M5 Blade Server.** See `https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-b-series-blade-servers/datasheet-c78-739296.html`. This server provides up to two Intel Xeon Scalable processors with up to 28 cores per CPU and up to 3.0 TB max. memory. △

2.2.1.2 Physical storage servers

A physical storage server houses the nonvolatile memory a consumer will need. This memory could be in the form of tapes (yes, they still exist, in a physical or virtual form), hard disks or Solid State Disks.

The physical connection of this memory to a compute server can take several forms: for example, Network Attached Storage (NAS, see `https://en.wikipedia.org/wiki/Network-attached_storage`) or Storage Area Network (SAN, see `https://en.wikipedia.org/wiki/Storage_area_network`). The "traditional" method employed a separate network connecting the storage servers to the compute servers; this network employed Infiniband or Fibre Channel protocols at layer 2. The switch fabric network that connected the compute servers utilized Ethernet at layer 2. A fairly recent trend, known as Hyper Converged Infrastructure (HCI), is to use Ethernet in the storage-compute server connection.

In a nutshell, for the hats of our interest, a *physical storage server provides consumers with one resource: bytes and bytes, lots of bytes of nonvolatile memory*. Like a desktop or laptop environment with directly attached storage, with the intervention

of the operating system, the raw storage is offered to the user in a virtualized form (e.g., block storage, a file system, a database).

Let's see some random examples next, in order to get an idea about the capabilities of some commercially available storage servers.

2.2.1.3 Switch Fabric

The Switch Fabric is the network that interconnects the compute servers inside the datacenter. Traditionally, its components were layer 2 switches, running Ethernet. The limitations of the STP protocol and the need to "isolate" consumers led to the components becoming routers - making the name Switch Fabric a misnomer for this network.

The topology of this network is highly "regular": Leaf and Spine or three layer (Access, Distribution and Core). The speeds of the links are increasing; the present maximums are 400Gbps per port.

In a nutshell, for the hats of our interest, the Switch Fabric is "invisible".

Example 2.4. **Arista Switch Fabric.** See `https://www.arista.com/en/products/converged-cloud-fabric` (also available as reference [12]). △

Example 2.5. **Juniper Switch Fabric.** Reference [4] provides a nice introduction to switch fabrics, if you are interested. It is also the entry point to start exploring Juniper's implementation. △

2.2.1.4 Gateways

These are routers that connect a datacenter to the "outside world"; that is, to consumers or other datacenters of the same or a different provider.

In a nutshell, for the hats of our interest, the gateways are "invisible".

2.2.2 Software components

This group contains all the software running in the hardware components we mentioned in the previous section.

2.2.2.1 Consumer applications

This component refers to software that consumers are running on the (compute) servers.

This component will be of interest to our hats, in several ways. A lot of business and technical requirements are centered around it, as we'll see in the next two chapters.

Example 2.6. **A simple, "stateles" (sstand alone) application.** Your own little program in your free AWS account running Matlab. It only needs a CPU resource in a server. When the application terminates, the system does not need to "save its state". △

Example 2.7. **A "stateful" application.** Such applications typically include databases. Enterprise applications like CRM, gmail, fall into this category. When the application terminates, the system needs to "save its state". △

Example 2.8. **An application that reports election results[5].** Consider a national election. At the end of the voting window, tallies from all voting stations are sent to a central location and current averages are reported on the web.

This application works for only a few hours. It accommodates two types of input: results from polling stations and requests from users anxious to see how their favorite party fared in the election. The first type of input "surges" when the polling stations close. The second one is rather unpredictable.

This application typically includes databases. △

Example 2.9. **Applications housed in multiple clouds.** In order to attract business, the "big 3" public clouds (AWS, Azure, GCP) (see Section 3.4.1, page 60) advertise differentiators - services in which they excel. For example, GCP claims excellence in analytics.

Multicloud applications aim to take advantage of these differentiators. Such applications run their logic in two or more clouds. They are the subject matter of the advanced course.

△

Example 2.10. **Moodle, the open source learning platform.** As professors, we are using Moodle at NC state, to communicate course content to students. As of May 24, 2023, according to `moodle.org`, 161,831 sites in 242 countries were using moodle to deliver 44,476,671 courses.

This is an impressive application. In Problem 2.13, we ask you to kick the tires a little more. If you are interested in contributing to the open source community, you may consider "Joining the Moodle Experience Lab", "Translating Moodle Academy" or "Experimenting with us", as described in `moodle.org`. △

2.2.2.2 Control software

This is software that runs on any hardware component in general. It is owned by the cloud provider.

[5] On May 21, 2023, when one of the authors was working on the second edition of these notes, national elections took place in Greece

Example 2.11. **Routing protocols.** This is software running on the routers and gateways. Examples of routing protocols used in switch fabrics are OSPF, BGP, and IS-IS. △

Example 2.12. **Switch fabric virtualization protocols.** The best known example of a switch fabric virtualization protocol used in switch fabrics is VxLAN. It was introduced in 2014 and described in RFC 7348. A competitor protocol is Geneve, introduced in November 2020 and described in RFC 8926. △

Example 2.13. **Linux, TCP, IP.** This (hopefully well-known) software runs on the compute and/or storage servers. △

2.2.2.3 Management software

This group includes applications that implement all management tasks. The applications run on any hardware component in general. They are owned by the cloud provider and/or consumer.

Example 2.14. **Server load balancers.** When more than one compute servers are available to process consumer requests, one must be chosen. The software that implements the selection logic is the server load balancer, a central piece in modern datacenters. Both the cloud provider as well as a consumer may run this software. △

Example 2.15. **Monitoring agents.** Run by provider or consumer; they are essential components for the SLAs, the agreements between providers and consumers (see Section 3.1.7.3, page 51) and any feedback-based control (e.g., Kubernetes, see Section 5.4, page 113.) △

Example 2.16. **VMotion.** We quote from [5] (the emphasis is ours): "...VMware's VMotion^{TM} is a key enabling technology for creating the dynamic, automated, and self-optimizing data center. It enables the live *migration of running virtual machines from one physical server to another* with zero downtime, continuous service availability, and complete transaction integrity."

The ability to move work around the compute servers is a great tool for the architects, as we'll see in a later chapter - and a great pain for another hat... △

2.3 Workloads and tenants in a datacenter

2.3.1 What is a workload?

Collectively, *the applications that users run on the servers are also known as the "workload" of the datacenter.* An application in the workload receives "work" from the users of the application - this is *requests* to be processed by the servers.

Example 2.17. **A bank application that processes ATM transactions.** This application receives data from an ATM location, queries a database, does some logging and sends a response back to the ATM. The work here is the individual requests that the ATMs send. Note that this application runs in two different servers: the database server and a "front end" that handles the remaining parts of the transaction.

It is conceivable that one of the two servers may be "overwhelmed" by the requests coming in. After all, a single server has a certain capacity to do work. What happens in the load exceeds that capacity?

One of the two big advantages of cloud computing (in our own opinion) is the ability of the cloud environment to increase this capacity by efficiently and quickly throwing more servers to handle a burgeoning workload[6]. △

2.3.2 Tenants in a datacenter

The notion of a *tenant* is an *administrative* one. In other words, what entity constitutes a tenant is arbitrary. Usually, tenants are defined in order to facilitate admin tasks. Loosely speaking, any subset of the applications that run in the datacenter can be considered a tenant.

Example 2.18. **Tenants in the datacenter of a bank.** All applications that belong to the HR department can be considered a tenant. The reason for that may simply be the requirement to manage the security of these applications in the same manner.

Another tenant may be all applications that generate monthly reports. △

Example 2.19. **Tenants in Facebook's datacenters.** Each Facebook user can be considered a tenant[7]; in this case, Facebook has to manage close to 2B tenants! △

2.3.3 Competition for resources

We refer to the servers, storage and switch fabric as resources. A precise definition and the rationale for this term will be presented later, when we discuss design problems.

Loosely speaking again, the applications in the workload running in a datacenter "compete" with each other for the use of resources. This is of course true for applications that belong to the same tenant as well.

Controlling how resources are allocated (i.e., how this competition is managed) has a profound effect on the way the datacenter operates and how the technical requirements mentioned in Section 6.2, page 149 are met.

---

[6] This is the elasticity characteristic of the cloud, as we'll discuss in Section 6.2, page 149.

[7] Since the notion of a tenant is arbitrary, this is the authors' hypothesis.

2.4 The problems of our interest

One can write a long list of problems for each one of the roles we mentioned in the previous chapter and for each one of the components of the datacenter.

In these notes, we'll not address any problems regarding the physical plant and hardware components. We have briefly mentioned them for completeness of exposition[8].

The focus of our interest in these notes and course will be on the control and management software mentioned in Sections 2.2.2.2 and 2.2.2.3.

We'll present the "headaches[9]" that issues in this area raise for the two technical roles we selected to discuss. In a more technical jargon, these headaches are the business requirements we outline in Chapter 4.

We'll then outline some "medicines for these headaches" - solutions to the issues in the following two chapters. Every medicine has side effects. The solutions have tradeoffs. In Chapter 6, we made a sincere effort to draw your attention to not only the solution, but also the tradeoffs the architect has to make. No free lunch in engineering...

[8] Operation of the physical plant is costly both in terms of $$ cost as well as environmental impact. It was deemed significant enough by Amazon to be included as the "sixth pillar" of their Well-architected" Framework in 2021; more on that in Section 4.3, page 83.

[9] and top $$

2.5 Problem Section

Problems on components of datacenters

Problem 2.1. Physical servers. Read the Wikipedia article on Servers `https://en.wikipedia.org/wiki/Server_(computing)` (also available as reference [13]).

1. Describe, in your own words, the difference between the client-server and publish-subscribe patterns.
2. Describe, in your own words, what database servers, file servers, mail servers, print servers, web servers, game servers, and application servers do.
3. The article does not mention storage servers. Google the term storage server; describe the similarities and differences between a compute and storage server.
4. Google "top server companies"; you'll get a variety of answers. That should be no surprise, the criteria for who's top are subjective. List 3-5 names of manufacturers that may appear frequently in such lists.
5. Pick such a name (e.g., HPE). Find out datasheets for servers. Get an idea of the state of the art features and capabilities.

Problem 2.2. Storage. Find out about the size of data stored worldwide. A starting point is
`https://www.statista.com/statistics/1062879/worldwide-cloud-storage-of-corporate`
Consult a few other sources, since these statistics are wild estimates.

1. What's the average of your findings?
2. How big is a petabyte? a zettabyte?

Problem 2.3. Storage. Nutanix (`https://www.nutanix.com/` is a manufacturer of storage products; they have a local, RTP presence and they hire NCSU graduates frequently. It might be a good idea to know the company a little better...

1. Find the model numbers of a few (hardware) storage devices with capacities over 1TB.
2. In `https://www.nutanix.com/sg/solutions`, they classify Storage as "Files Storage", "Objects Storage", and "Volumes Block Storage". Describe, in your own words, these types of storage.

Problem 2.4. Switch Fabric. Read `https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/`; it describes the technology used in the switch fabrics inside Facebook's datacenters. Get an idea of the state of the art features and capabilities.

1. Summarize, in your own words, features that attracted your attention.

Problems on datacenter applications, workloads and tenants

Problem 2.5. Consider the election result reporting application in Example 2.8, page 30.

Suppose that this application is used to report NC state election results. *Make " reasonable assumptions for this scenario.*

1. Who could be running this application?
2. How many (result reporting) input streams are there?
3. Describe some requirements you would impose on how a polling station reports its results to the application.
4. Can you provide an estimate about the bandwidth needed to collect all results?
5. Who could be accessing this application for viewing the results?
6. Can you provide an estimate about the bandwidth needed to support viewers?
7. Can you provide an estimate about the memory needed to store the results?
8. How can a viewer "discover" the application?
9. How can a polling station "discover" the application?
10. What security concerns can you raise about this application? Do not worry about how one can address the concerns.
11. (Optional) Regardless of the reason, if the application "crashes" in the middle of election night, the IT person in charge will, in all likelihood, lose their job. What would you do, if you were that person, to not lose your job?

Problem 2.6. Consider the moodle application we mentioned in Example 2.10, page 30. Statistics about sites, courses and countries are mentioned in `moodle.org`. How, in your opinion, are the statistics calculated? More specifically,

1. what are the required inputs?
2. where are these inputs generated?
3. where are they processed?
4. how are they transferred there?

Problem 2.7. Consider `https://www.nfl.com/`, the official website of the NFL.

1. Describe as many types of requests as you can, for this website.
2. For each type, mention very clearly, who can potentially create such requests.
3. For one type only, can you hypothesize about the "pattern" of such requests (e.g., it is uniform, may have peaks, etc.)?
4. For one type only, can you guesstimate how much time a request would take to be processed? State your assumptions clearly.

Problem 2.8. Applications with variable memory requirements. An application requires two types of (main) memory. The *instruction* part is where the instructions are stored and the *data* part is used for calculations, entering input data, etc.

1. Describe several reasons that make these parts grow in size, as the application runs. Comment on the nature of these reasons (e.g., random, programming errors, etc.).
2. Describe how a typical Operating System deals with such growth demands.

3. Describe how a typical programming language deals with such growth demands.
4. What happens if the application outgrows the memory capacity of the system it is running on?

In Chapter 5, we'll see how an application's needs for memory can be addressed.

Problem 2.9. Workload in Google's public DNS server. Google provides a public, free DNS service (IP addresses 8.8.8.8 and and 8.8.4.4). According to some estimates, the server handles 1 trillion requests per day.

1. Describe, in your own words, what is a DNS request.
2. Assume that the requests hit the DNS server at a uniform rate. What is this rate in requests per second?
3. Google "average DNS lookup time" to find out how much time a request to resolve a name would take. In your opinion, what are the components of this delay?
4. Make a reasonable guess about the time the DNS server spends in processing a request. State your assumptions clearly.
5. What is the "magnitude of the load" on this server?
6. Is one instance of the server sufficient to handle the load? Explain.
7. Take a look at namebench, Google's Open Source DNS Benchmark Utility; you can find it at `https://code.google.com/archive/p/namebench/` (also available as reference [15]). Run it and see how close your guess and this benchmark's output were.
8. Assume that requests come in bursts, instead of uniformly. Figure 2.1 gives you an idea. Assume that the average rate is given by the dotted, horizontal line.

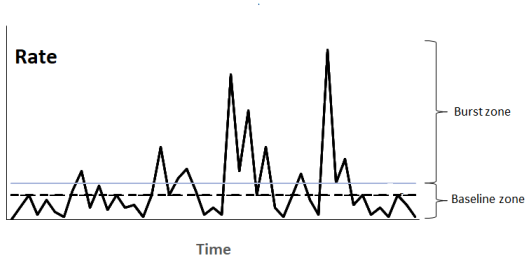


Fig. 2.1: Bursty pattern of DNS requests.

What is the "magnitude of the load" on this server in the bursty requests scenario?

Problem 2.10. Workloads in a Google datacenter. Find out articles for the types of workloads Google runs in their datacenters (index calculation, gmail, drive, etc.). Describe, in your own words, the main features of these workloads.

Problem 2.11. Workloads in a Facebook datacenter. Find out articles for the types of workloads Facebook runs in their datacenters. Describe, in your own words, the main features of these workloads.

Problem 2.12. Workloads in an AWS datacenter. Find out articles for the types of workloads Amazon hosts in their datacenters. Describe, in your own words, the main features of these workloads.

Problem 2.13. Consider the moodle application we mentioned in Example 2.10, page 30. As mentioned, 161,831 sites are using the application. In this problem, we focus only on NC State's site.

1. Can a specific course like ECE/CSC547 be considered a tenant in NC State's moodle site? Why?
2. Can a specific student you be considered a tenant? Why?
3. Can you specify a third type of tenant?

Problem 2.14. Tenants in a Facebook datacenter. Consider the set of Facebook users. Supply at least three different criteria for partitioning this set into tenants. Describe how you could identify traffic entering a datacenter as belonging to a specific tenant. Discuss, if possible, advantages and disadvantages of your proposal.

Problem 2.15. Tenant competition in a Facebook datacenter. Consider the set of Facebook users. Suppose, for simplicity, that each individual user is a tenant.

1. Describe how users compete for resources.
2. Describe how you could resolve this competition. Discuss, if possible, advantages and disadvantages of your proposal.

Huh?

Problem 2.16. How big is the volume of "big data". Nowadays, big data is a big buzzword."Data is the new oil" is perhaps one of the most popular catchphrases highlighting the importance of data. Everyone is looking to analyze it and profit from it. It appears that no one is paying attention to a lowly housekeeping function. So, let us take a look at it.

Continuing in the spirit of Problem 2.2, read [11]. According to this source,

• The amount of data in the world was estimated to be 44 zettabytes at the dawn of 2020.

• By 2025, the amount of data generated each day is expected to reach 463 exabytes globally.

1. Find out what the volume of a SSD (Solid State Disk) is.
2. How much datacenter space in cubic meters would a zettabyte consume? State your assumptions clearly.
3. Citadel, a datacenter located in Tahoe Reno, Nevada, covers an area of 7.2 million square feet. It is considered one of the largest datacenters in the world[10]. Find out the *volume* of the Citadel.

---

[10] It topped the "8 Largest Data Centers in the World" in 2020.

4. Can we store all data the world had at the dawn of 2020 in Citadel? State your assumptions clearly.
5. Repeat the last question for year 2025.
6. Assume linear growth of data in the future. When will we cover the island of Ibiza, Spain with datacenters that just store data? State your assumptions clearly.

References

References marked as Handout H2.x are required reading material; you will find them in the required handouts binder in moodle. The rest are used in some form in the text or problems.
The references on physical servers are for your own use, in case you are interested.

1. **Handout H2.1:** Brian Daigle, "Data Centers Around the World: A Quick Look" https://www.usitc.gov/publications/332/executive_briefings/ebot_data_centers_around_the_world.pdf
2. **Reference R2.1:** Wikipedia article on Datacenters https://en.wikipedia.org/wiki/Data_center
3. **Reference R2.2:** The 10 largest datacenters in the world, https://analyticsindiamag.com/10-largest-data-centres-in-the-world/
4. **Reference R2.3:** Juniper short note: What is a data center fabric? https://www.juniper.net/us/en/research-topics/what-is-data-center-fabric.html
5. **Reference R2.4:** VMotion datasheet, https://www.vmware.com/pdf/vmotion_datasheet.pdf
6. **Reference R2.5:** Dell technologies, datasheet of a random server https://www.delltechnologies.com/asset/en-us/products/servers/briefs-summaries/dell-emc-poweredge-15g-portfolio-brochure.pdf
7. **Reference R2.6:** Lenovo, datasheet of a random server https://lenovopress.com/lp1045-thinksystem-sr530-server
8. **Reference R2.7:** Oracle, datasheet of a random server https://www.oracle.com/a/ocom/docs/servers/sparc/sparc-m8-8-ds-3864183.pdf
9. **Reference R2.8:** HPE, datasheet of a random server https://www.hpe.com/psnow/doc/a00008159enw.pdf
10. **Reference R2.9:** Cisco, datasheet of a random server https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-b-series-blade-servers/datasheet-c78-739296.pdf
11. **Reference R2.10:** How Much Data Is Created Every Day? [27 Staggering Stats], https://seedscientific.com/how-much-data-is-created-every-day/
12. **Reference R2.11:** Arista Switch Fabric, https://www.arista.com/en/products/converged-cloud-fabric
13. **Reference R2.12:** Wikipedia article on Servers, https://en.wikipedia.org/wiki/Server_(computing)
14. **Reference R2.13:** Reinventing Facebook's data center network, https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/
15. **Reference R2.14:** Google's Open Source DNS Benchmark Utility, https://code.google.com/archive/p/namebench/
16. **Reference R2.15:** Lights-out data center, https://www.datacenterdynamics.com/en/analysis/what-lights-out-data-center/

Topic 3
Cloud computing analysis

Objectives: After reading this chapter, you should be able to: (a) describe the notion of service, (b) explain the differences in the ways a cloud is deployed, and, (c) analyze the most common cloud platforms (SaaS, IaaS, PaaS, FaaS) and explain the main differences.

The "meat" of these notes for the architect hat is Chapter 6. Good system design (aka "synthesis") is empowered by the ability to "analyze" existing systems. In this chapter, we focus on analyzing cloud computing systems. We first identify some notions that are fundamental for the analysis and then use them to provide a taxonomy of the existing cloud offerings. This will provide the background for describing the problems the architect is called to solve.

3.1 Fundamental notions in analysis of a cloud

In this section, we discuss briefly some notions that we consider fundamental. These are concepts or technologies that appear in several components of a cloud.

3.1.1 The notion of virtualization

Virtualization is everywhere, in cloud environments. Arguably[1], virtualization is **the single most important concept** in cloud computing. Its applications appear in all

[1] We say arguably, because the **service** concept is a very strong competitor. After all, service wins, hands down, in the name recognition contest.

technologies use in cloud computing - servers, storage, and networks. We emphasize the topic of server virtualization in this chapter.

We provide an overview of some basic concepts that help in understanding its application in such different technologies in [4].

What is common in all these applications? How can we understand it?

Virtualization is the process of giving a **system** a property it **does not have**.

The key words in this abstract definition are in boldface. In studying all the implementations of virtualization in cloud computing, three questions should be answered:

- Which hat is responsible for giving the missing property?
- Which hat benefits from the new property?
- Which hat pays the price, since there is no free lunch in engineering?

3.1.2 Server virtualization

We quote from [3]:

What is Server Virtualization?

Server virtualization is used to mask server resources from server users. This can include the number and identity of operating systems, processors, and individual physical servers.

Server Virtualization Definition

Server virtualization is the process of dividing a physical server into multiple unique and isolated virtual servers by means of a software application. Each virtual server can run its own operating systems independently.

Key Benefits of Server Virtualization:

• Higher server ability
• Cheaper operating costs
• Eliminate server complexity
• Increased application performance
• Deploy workload quicker

Three Kinds of Server Virtualization:

• **Full Virtualization:** Full virtualization uses a hypervisor, a type of software that directly communicates with a physical server's disk space and CPU. The hypervisor monitors the physical server's resources and keeps each virtual server independent and unaware of the other virtual servers. It also relays resources from the physical server to the correct virtual server as it runs applications. The biggest limitation of using full virtualization is that a hypervisor has its own processing needs. This can slow down applications and impact server performance.

• **Para-Virtualization:** Unlike full virtualization, para-virtualization involves the entire network working together as a cohesive unit. Since each operating system on the virtual servers is aware of one another in para-virtualization, the hypervisor does not need to use as much processing power to manage the operating systems.

• **OS-Level Virtualization:** Unlike full and para-virtualization, OS-level visualization does not use a hypervisor. Instead, the virtualization capability, which is part of the physical

server operating system, performs all the tasks of a hypervisor. However, all the virtual servers must run that same operating system in this server virtualization method.

Why Server Virtualization? Server virtualization is a cost-effective way to provide web hosting services and effectively utilize existing resources in IT infrastructure. Without server virtualization, servers only use a small part of their processing power. This results in servers sitting idle because the workload is distributed to only a portion of the network's servers. Data centers become overcrowded with underutilized servers, causing a waste of resources and power.

By having each physical server divided into multiple virtual servers, server virtualization allows each virtual server to act as a unique physical device. Each virtual server can run its own applications and operating system. This process increases the utilization of resources by making each virtual server act as a physical server and increases the capacity of each physical machine.

3.1.3 Virtual machines

We quote from [5]:

What is a virtual machine? A **Virtual Machine (VM)** is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual "guest" machines run on a physical "host" machine. Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host. This means that, for example, a virtual MacOS virtual machine can run on a physical PC.

Virtual machine technology is used for many use cases across on-premises and cloud environments. More recently, public cloud services are using virtual machines to provide virtual application resources to multiple users at once, for even more cost efficient and flexible compute.

What are virtual machines used for?

Virtual machines (VMs) allow a business to run an operating system that behaves like a completely separate computer in an app window on a desktop. VMs may be deployed to accommodate different levels of processing power needs, to run software that requires a different operating system, or to test applications in a safe, sandboxed environment.

Virtual machines have historically been used for server virtualization, which enables IT teams to consolidate their computing resources and improve efficiency. Additionally, virtual machines can perform specific tasks considered too risky to carry out in a host environment, such as accessing virus-infected data or testing operating systems. Since the virtual machine is separated from the rest of the system, the software inside the virtual machine cannot tamper with the host computer.

How do virtual machines work?

The virtual machine runs as a process in an application window, similar to any other application, on the operating system of the physical machine. Key files that make up a virtual machine include a log file, NVRAM setting file, virtual disk file and configuration file.

Advantages of virtual machines

Virtual machines are easy to manage and maintain, and they offer several advantages over physical machines:

• VMs can run multiple operating system environments on a single physical computer, saving physical space, time and management costs.

• Virtual machines support legacy applications, reducing the cost of migrating to a new operating system. For example, a Linux virtual machine running a distribution of Linux as

the guest operating system can exist on a host server that is running a non-Linux operating system, such as Windows.

• VMs can also provide integrated disaster recovery and application provisioning options.

Disadvantages of virtual machines While virtual machines have several advantages over physical machines, there are also some potential disadvantages:

• Running multiple virtual machines on one physical machine can result in unstable performance if infrastructure requirements are not met.

• Virtual machines are less efficient and run slower than a full physical computer. Most enterprises use a combination of physical and virtual infrastructure to balance the corresponding advantages and disadvantages.

The two types of virtual machines Users can choose from two different types of virtual machines—process VMs and system VMs:

A **process virtual machine** allows a single process to run as an application on a host machine, providing a platform-independent programming environment by masking the information of the underlying hardware or operating system. An example of a process VM is the Java Virtual Machine, which enables any operating system to run Java applications as if they were native to that system.

A **system virtual machine** is fully virtualized to substitute for a physical machine. A system platform supports the sharing of a host computer's physical resources between multiple virtual machines, each running its own copy of the operating system. This virtualization process relies on a hypervisor, which can run on bare hardware, such as VMware ESXi, or on top of an operating system.

What are 5 types of virtualization? All the components of a traditional data center or IT infrastructure can be virtualized today, with various specific types of virtualization:

• **Hardware virtualization:** When virtualizing hardware, virtual versions of computers and operating systems (VMs) are created and consolidated into a single, primary, physical server. A hypervisor communicates directly with a physical server's disk space and CPU to manage the VMs. Hardware virtualization, which is also known as server virtualization, allows hardware resources to be utilized more efficiently and for one machine to simultaneously run different operating systems.

• **Software virtualization:** Software virtualization creates a computer system complete with hardware that allows one or more guest operating systems to run on a physical host machine. For example, Android OS can run on a host machine that is natively using a Microsoft Windows OS, utilizing the same hardware as the host machine does. Additionally, applications can be virtualized and delivered from a server to an end user's device, such as a laptop or smartphone. This allows employees to access centrally hosted applications when working remotely.

• **Storage virtualization:** Storage can be virtualized by consolidating multiple physical storage devices to appear as a single storage device. Benefits include increased performance and speed, load balancing and reduced costs. Storage virtualization also helps with disaster recovery planning, as virtual storage data can be duplicated and quickly transferred to another location, reducing downtime.

• **Network virtualization:** Multiple sub-networks can be created on the same physical network by combining equipment into a single, software-based virtual network resource. Network virtualization also divides available bandwidth into multiple, independent channels, each of which can be assigned to servers and devices in real time. Advantages include increased reliability, network speed, security and better monitoring of data usage. Network virtualization can be a good choice for companies with a high volume of users who need access at all times.

• **Desktop virtualization:** This common type of virtualization separates the desktop environment from the physical device and stores a desktop on a remote server, allowing users to access their desktops from anywhere on any device. In addition to easy accessibility,

benefits of virtual desktops include better data security, cost savings on software licenses and updates, and ease of management.

Container vs virtual machine Like virtual machines, container technology such as Kubernetes is similar in the sense of running isolated applications on a single platform. While virtual machines virtualize the hardware layer to create a "computer," containers package up just a single app along with its dependencies. Virtual machines are often managed by a hypervisor, whereas container systems provide shared operating system services from the underlying host and isolate the applications using virtual-memory hardware.

A key benefit of containers is that they have less overhead compared to virtual machines. Containers include only the binaries, libraries and other required dependencies, and the application. Containers that are on the same host share the same operating system kernel, making containers much smaller than virtual machines. As a result, containers boot faster, maximize server resources, and make delivering applications easier. Containers have become popular for use cases such as web applications, DevOps testing, microservices and maximizing the number of apps that can be deployed per server.

Virtual machines are larger and slower to boot than containers. They are logically isolated from one another, with their own operating system kernel, and offer the benefits of a completely separate operating system. Virtual machines are best for running multiple applications together, monolithic applications, isolation between apps, and for legacy apps running on older operating systems. Containers and virtual machines may also be used together.

Setting up a virtual machine Virtual machines can be simple to set up, and there are many guides online that walk users through the process. VMware offers one such useful virtual machine set-up guide.

3.1.4 Containers

We quote from [6]:

What are Containers? A **Container** is a lightweight, standalone package that encapsulates a complete runtime environment including an application and its dependencies (libraries, binaries, and any additional configuration files), increasing an application's portability, scalability, security, and agility.

Containers are popular with both developers and operators because they offer a straightforward way to deploy and manage applications, regardless of the target environment. They facilitate DevOps and DevSecOps practices by improving handoffs between development and operations teams.

Containers consume resources efficiently, enabling high density and resource utilization. Although containers can be used with almost any application, they're frequently associated with microservices, in which multiple containers run separate application components or services. The containers that make up an application are typically coordinated and managed using a container orchestration platform, such as Kubernetes.

Containers are created by packaging applications from multiple images from one or more repositories along with any libraries or other application dependencies, eliminating portability and compatibility issues.

Containers vs. VMs

Each virtual machine (VM) runs a full or partial instance of an operating system, whereas multiple containers share a single operating system instance. A container includes everything necessary to run an application, enabling multiple containerized applications to run independently on a single host system. Because multiple containers can be run inside a VM, the two technologies can be used together

What are the benefits of using containers? Using containers to build applications accelerates the delivery of new functionality and encourages an environment of continuous innovation. Benefits include:

Agility. Improved developer agility drives increased productivity and the speed of app development. Containers streamline CI/CD pipelines and are ideal for DevOps teams and microservices deployments.

Scalability and **high availability.** Using Kubernetes, container deployments can automatically be scaled up or down as workload requirement changes, increasing app availability.

Portability. Containers consume fewer resources and are lighter weight than VMs. Containerized applications are infrastructure-agnostic and operate the same regardless of where they are deployed.

Resiliency. A containerized application is isolated and abstracted from the OS and other containers; one container can fail without impacting other running containers.

Container standards Standards for container formatting and runtime environments are controlled by the Open Container Initiative (OCI), a project formed in 2015 for the express purpose of creating open industry standards. The OCI currently offers two specifications: the Runtime Specification (runtime-spec) and the Image Specification (image-spec).

Container security Containers require changes to the way security policies are implemented and managed. Security should be built into the container lifecycle as much as possible, using a DevSecOps approach. Security teams, working with development and operations teams, adapt existing governance and compliance policies to accommodate new tools and changes to the application lifecycle.

Container automation Manual effort slows down development teams. Container automation enables developers to focus on code instead of packaging. Container images are built in layers. With an automated approach to container builds, whenever a layer changes, only that layer has to be updated. For example, if only system libraries need to be updated, only the layer containing the libraries must be rebuilt. Because other layers remain unchanged, the testing and validation burden is reduced, enabling updated containers to be pushed into production faster and more frequently.

Docker Since its introduction in 2013, Docker has been almost synonymous with containers, and it continues to be used to build container images. The Docker environment includes a container runtime as well as container build and image management. Because Docker builds an OCI-standard container image, Docker images will run on any OCI-compliant container runtime.

Kubernetes Kubernetes is an open source container orchestrator that has become a de facto standard. Kubernetes automates deployment, load balancing, resource allocation, and security enforcement for containers via declarative configuration and automation. It keeps containerized applications running in their desired state, ensuring they are scalable and resilient. Container orchestration helps manage the complexity of the container lifecycle. This becomes especially important for distributed applications with large numbers of containers.

Kubernetes is shifting to the Container Runtime Interface (CRI), which supports a broader set of container runtimes with smooth interoperability between different runtimes. Deprecation of the Docker runtime for use with Kubernetes was announced in December, 2020.

Container use cases

For developers

• Improve application portability across different platforms and configurations, so that code developed on one version of a language compiler or interpreter runs flawlessly on subsequent versions with no revisions required.

• Free developers from having to develop, test, and deploy on the same infrastructure, so that developers who write code on their laptops can be confident that application will run as desired on any other infrastructure, whether on-premises server or a cloud-based VM.

• Facilitate agile development processes such as CI/CD, speeding code acceptance and deployment.

For IT operations

• Improve application security by isolation from other applications in a lightweight fashion.

• Seamless migration of containerized applications across different OS versions, network topologies, or storage configurations, and cloud platforms.

• Improve IT efficiency by enabling multiple application containers to run on a single OS instance. Since containers are often tens of megabytes in size where VMs are often ten or more gigabytes in size, a substantially larger number of containers can run on a single server instance.

• Deliver extreme on-demand scalability, by spinning up additional container instances in milliseconds versus minutes to spin up VMs.

3.1.5 Containers vs VMs

We quote from [7]:

Why use containers vs. VMs? Virtual Machines (VMs) and Containers are complimentary (*sic*) and similar – both improve IT efficiency, application portability, and enhance DevOps. However, understanding the difference between them is a key component of developing an agile, cloud-native, application-driven strategy.

• VMs solve infrastructure problems by letting organizations get more out of servers and facilitate limited workload portability.

• Containers solve application problems by improving DevOps, enabling microservices, increasing portability, and further improving resource utilization.

What is the core difference between containers and VMs? VMs include the guest operating system (OS) along with all the code for their applications and application dependencies that formerly ran on a single server or from a server pool. The size of VM images is generally measured in gigabytes. Multiple VMs can exist on a single physical server even if they are running on different operating systems. VMs abstract servers from the underlying hardware and typically persist throughout their useful life.

Containers share the host OS and include only the applications and their dependencies. The size of container images is generally measured in megabytes. Every container running on a single server shares the same underlying OS. Containers thus can spin up in milliseconds and are more efficient for ephemeral use cases where instances must be spun up and down with changes in demand.

In summary, how are VMs and containers different? Although both containers and VMs help improve the utilization of IT resources, each has its pros and cons. Virtual machines have been around for decades, allowing enterprises to combine several servers running different applications onto a single physical server, even if they run different operating systems. This has enabled substantial savings in server hardware and software as what used to run on several servers now runs on a single server instead. VMs are also the underpinnings of most cloud services. AWS, Azure, and other public clouds use the VM as one of their standard offerings. However, since they encapsulate an entire server in each virtual machine, the amount of CPU and RAM that VMs require can become unwieldly and limit the number of VMs that can exist on a single server.

Containers have rapidly gained popularity since the release of Docker in 2013, partly as a response to the amount of overhead that VMs consume. Since containers ride on a server's OS, they share that single OS instance and other binaries and libraries, so containers need to only include application code, whether in the form of a single monolithic application or

microservices that are bundled together in one or more containers to encompass a business function.

Thus, while VMs let an organization run several virtual servers on a single piece of hardware – regardless of their operating systems, containers offer lightweight, high-density application virtualization, the ability to spin applications and instances up and down in seconds, and some measure of security inherent in separating applications in their own containers.

What are the pros and cons of virtualization?

VM Pros:

• Decades of virtualization expertise enables access to a robust set of VM management and security tools

• VMs offer the ability to run multiple applications requiring different OSs on a single piece of infrastructure

• VMs emulate an entire computing environment, including all OS resources

• VMs simplify the portability and migration between on-premises and cloud-based platforms

• There is a vast, established VM ecosystem and marketplace with industry leaders such as VMware

VM Cons:

• VM images typically consume gigabytes and thus take longer to backup or migrate between platforms

• Because they encapsulate the entire server including OS, a physical server can support fewer VMs than containers

• VM Spin-up time can take minutes

What are the pros and cons of containers?

Container Pros:

• Containers are more lightweight than VMs, as their images are measured in megabytes rather than gigabytes

• Containers require fewer IT resources to deploy, run, and manage

• Containers spin up in milliseconds

• Since their order of magnitude is smaller

• A single system can host many more containers as compared to VMs

Container Cons:

• All containers must run atop the same OS – no mix and match of OSs or versions

• Containers may be less secure than VMs since the underlying OS is shared

• Containers are a newer technology, and the ecosystem is still evolving

How are containers used in traditional vs. emerging IT practices? VMs have and will continue to play a role in migrating legacy applications to the cloud and hybrid environments. Due to VM maturity there is a high comfort level as well as a level or inertia for those organizations that have settled into an IT architecture that revolves around VMs. As a result, VMs will continue to be the abstraction solution of choice for many persistent, monolithic, enterprise applications, especially those applications that do not have frequent updates.

However, containers lend themselves to modern practices and uses cases, such as CI/CD in agile, DevOps environments. Containers enhance portability of apps between vastly different configurations, so an application developed on a laptop and tested in a sandbox can run in the cloud with no changes required to support all three environments. Containers also offer near limitless scalability. Microservices-based applications that separate user interaction from back-end processing let front and back end each scale separately, and containers light weight mean nearly instantaneous spin-up of new instances as needed. Furthermore, taking a microservices approach to development encourages sharing common microservices routines between multiple applications and business processes, further improving developer efficiency.

How is the role of VMs transitioning in emerging IT practices? As machine learning and artificial intelligence (ML/AI) applications permeate deeper into enterprises, these resource-heavy applications will favor VMs for deployment. Additionally, new network architectures like 5G which require more computing power at the edge will favor VMs to do the heavy lifting of these software-defined networks.

Containers will thrive in the world of modern, customer-facing applications and web services that require scalability, thanks to their near-instantaneous start-up time, and containers exceed as a development platform, since coders need to no longer worry about how development infrastructure varies from deployment infrastructure.

Finally, since containers were originally designed to be transient, they lend themselves well to network daemons, caching, and web services functions.

Will containers replace VMs, or do they complement each other? Both containers and virtual machines will continue to play important roles. Containers can run on VMs, enabling an organization to leverage its existing tools for automation, backup, and monitoring. Containers on VMs enable IT to use existing VM-savvy teams to manage a containerized environment as well. VMs will have new use cases as enterprises seek to leverage the power of their infrastructure – or the cloud – in new ways to support heavy-duty application and networking workloads.

3.1.6 Network virtualization

Virtualization appears in numerous places in networking as well. Inside a datacenter, switch fabric virtualization has solved numerous problems for the cloud provider administrators - as we explain in another course.

Example 3.1. **LAN virtualization.** In the LAN space, the first and most widely known example of Virtualization is the notion of VLANs. The new property is the decreased size of broadcast domains.

Another example is Etherchannel. The new property is increased bandwidth between two switches.

Yet a third one is VPLS. The new property is increased geographical area that the LAN spans. △

3.1.7 Services

Take a moment and, without reading further, answer the question yourself: “What is a service”?

The notion of service is ubiquitous in the entire IT world, not just cloud computing; it is also widely used in the economy (e.g., financial services, transportation services, tax preparation services) and social environments (e.g., childcare services). Understanding it in a more abstract way is, therefore, fundamental.

3.1.7.1 What is a Service?

Let's start with some intuitive descriptions in a nontechnical setting first.

Service is *something* that meets a need or fulfills a demand; Service is the action of helping or doing work for someone. Example: "Voluntary service at a soup kitchen".

Service is a *system* supplying a (public) need (such as transport, communications, or utilities such as electricity and water). Example:"a regular bus service".

It should be apparent that *two actors* are involved in a service: one who *gives* and one who *receives* the outcome of the service.

Let's jump now to a technical definition. The ITIL Practitioner Guidance publication [11] offers up the following definition (the emphasis is ours):

> "A service is a means of delivering value to customers by facilitating outcomes that customers want to achieve **without the ownership of specific costs and risks**."

This is quite generic. Let's specialize it in the context of cloud computing. The deliver*ers* are the providers; the deliver*ees* are the consumers. The consumers offload management tasks (and risks) to the providers.

What exactly is the abstract *value* that's delivered? The **use** of three "things" (*not the things themselves*): infrastructure, platforms or applications.

How? The provider specifies a *software interface* to the consumer; the consumer uses a *network* to access the "thing" they'll be using.

Putting all the emphasized terms together,

> "A cloud computing service is the use of infrastructure, platforms or applications by a consumer that is free of most management burdens. The use takes places over a network, via defined software interfaces."

3.1.7.2 Benefits and tradeoffs

The generic benefits for the recipient of a service are spelled out in the above definition: reduced costs and risks. We'll see later how these generic benefits translate into more precise technical terms.

Example 3.2. **SaaS benefits.** In Section 3.7, one benefit SaaS gives to the user is reduction of management tasks. For example, the user does not have to maintain the servers housing the software application. △

There must be benefits to the provider of the service as well, even though the definition does not explicitly spell them out. However, the use of the word "customer" provides a hint: profit. See [11] for a great, more detailed description of benefits.

As is the case with everything in engineering, there are tradeoffs associated with the design of (technical) services. We address them in the next chapters.

3.1.7.3 How are Services discovered and delivered

How a service is designed and implemented is, of course, quite important to the service provider; not so much so, to the consumer.

Three aspects are important to the consumer: (a) learn what services are available for her/his needs (**service discovery**), (b) determine how the service is accessed (**service delivery**), and, (c) understand what the provider can guarantee about the service.

In the case of software services, service delivery is done via the celebrated notion of *APIs, Application Programming Interfaces*.

The guarantees regarding a given service are outlined by the provider in legal, binding documents called Service Level Agreements (SLAs).

You can read the details of a typical SLA in the next example. A common characteristic of such SLAs is that they guarantee availability, not performance. If the provider messes up, the consumer gets a discount in the next billing cycle, that's all. We'll explain why SLAs have such a structure in the next chapter, once we introduce requirements more formally.

Example 3.3. **Azure SLA for using Kubernetes.** For details, see [17]. △

3.1.7.4 Service mesh

—— This section is a stub for this semester. ——

We quote from [8]

> **What is a Service Mesh?** A **service mesh** refers to the way that software code from cloud hosted applications is woven together at different levels of the webserver in integrated layers. Rather than functioning in an isolated runtime at the top layer of a web server stack configuration, cloud hosted application code can be built with APIs that facilitate calls to other software-driven services available at the level of the operating system, web server, network, or data center. A service mesh increases the potential functionality of software applications by extending the levels of interoperable communication between infrastructure elements in production.
>
> A service mesh weaves together thousands of microservices across VMs in an elastic cloud data center through automated, cross-channel communication between running applications. Dedicated service-to-service communication functionality is required by cloud orchestration, load balancing, resource discovery SDN routing, API communication, database synchronization, and script optimization applications across all levels of data center operations. A service mesh can be used for better data analytics and traffic metrics for multi-tiered network architecture across millions of multi-tenant rack servers at a time.
>
> **Benefits of a Service Mesh**
>
> • Increased interoperability: A service mesh extends the functionality of SDN routing features for integrated microservice environments in support of web, mobile, and SaaS application code.

• Enhanced microservice discovery: A service mesh improves network configuration and management through better microservice discovery.

• Detailed real-time monitoring and analytics of network activity: A service mesh can access backend processes and web server hardware for more detailed real-time monitoring and analytics of network activity.

• Powerful automation of web and mobile scripts: Developers can script service mesh functionality through YAML files or utilities like Vagrant, Jenkins, Puppet, Chef, etc. to build powerful automation of web and mobile scripts at scale. This type of architecture is required for supporting complex SaaS applications in production in enterprise. A service mesh provides coordination for thousands or millions of containers running simultaneously in the cloud.

How does a Service Mesh work? A service mesh works through discovery and routing applications that are installed on every VM instance or node in an elastic web server network to register running microservices by IP address. A central registry is used for the configuration, management, and administration of all the simultaneously running microservices on a network. The service mesh can be referenced by parallel applications operating at the various layers of a web server, data center, or application to extend interoperable functionality through data analytics and network monitoring. This leads to increased data center automation at the level of IP routing, SDN definitions, firewall settings, filters, rules, and cloud load balancing.

API connections can reference the service mesh for definitions of where to discover running applications and microservice features for data transfers or required processing activity. Elastic web server platforms that scale automatically with Kubernetes use Istio as the central registry and configuration management utility for microservice discovery. Elastic web server platforms like AWS EC2 and Kubernetes utilize the service mesh for managing multiple copies of cloud applications in simultaneous runtimes while synchronizing changes to master database and storage information. A service mesh permits the application layer to communicate with the webserver, internet, and data center network resources through APIs, or vice versa, depending on the requirements of the microservice or code base.

Service Mesh architecture A service mesh is based on an abstraction layer that is installed across VMs or containers in a cloud data center. Code is installed on every VM or node which communicates with a central administration software instance running the data center orchestration. Service mesh solutions like VMware NSX and Istio rely on Envoy to create the data plane at the node level. Envoy manages information related to the running microservices, licensed IP addresses, HTTPS encryption, active database formats, etc. for every VM or node. With NSX this includes distributed firewall integration at the level of the hypervisor. In elastic cloud networks, the data plane information for each VM or node is used for load balancing. API connections rely on service mesh architecture for inter-application routing requirements. Telemetry at Level 7 of the service mesh includes DNS, HTTP/S, SMTP, POP3, FTP, etc.

Service Mesh implementation The service mesh implementation includes load balancing and service discovery across the SDN, IP address, Microservice, and API resources of a web/mobile application. The service mesh manages communication, synchronization, and encryption for connections in the webserver backend across hardware in an elastic web server architecture. In cloud applications, the script, database, and static web files are often separated on different hardware, then assembled on the final page of the web browser.

The SDN routing between hardware, scripts, database, and files becomes even more complex with third-party APIs in the code. When all of this must be assembled across resources for every page load, the service mesh integrates, synchronizes, and standardizes the operation across VMs in elastic web server frameworks. The service mesh was created to meet a need that no other software provides in the data center. It also includes data analytics and user metrics from web traffic connections.

Open source Service Mesh Istio is currently the most advanced open source service mesh project, with Envoy being used for the central features related to the management of the data plane across nodes. Istio was originally developed as part of the Cloud Native Computing Foundation (CNCF) and works within the VMware NSX Service Mesh and Enterprise PKS platforms. PKS is VMware's Kubernetes distribution which orchestrates cloud web servers through containers. PKS is available as a self-hosted package for public and private cloud requirements or as a fully managed Containers-as-a-Service (CaaS) product. Istio is used for microservice communication in Kubernetes with complex IP address routing capabilities and encryption for elastic web server orchestration in enterprise data centers at scale. Linkerd, Conduit, Aspen, and Consul are other important open source projects being developed as components of service mesh frameworks.

Elastic Service Mesh An elastic service mesh is required to synchronize database and website files in a cloud hosting framework like AWS EC2 or Kubernetes. The service mesh controls the routing between VMs in the webserver backend for API and SDN requirements in software application support. When the service mesh is also used for discovery and load balancing in elastic web server networks, administrators can automate the allocation of data center resources to match the requirements of user traffic in production. Web servers can be configured to automatically launch or be terminated when no longer required for more efficient use of cloud hardware resources. The ability to embed real-time monitoring and analytics capabilities into a service mesh at the level of the VM or node provides software developers, programmers, and web publishers with the ability to create new features for applications using microservices.

Why microservices architecture needs a Service Mesh A public cloud may contain millions of simultaneously running microservices across containers or virtual machines supporting different applications and databases in parallel through isolated runtimes. Multitenant environments based on virtualization require a better method to discover and register microservices so that the unique functionality of each can be integrated by applications or shared to other devices using APIs. Many microservice formats are not designed specifically for elastic web server platforms and need a service mesh to manage the operation in containers. A service mesh provides the fine-grained routing and encryption functionality over SDN that allows different APIs to communicate between running code processes on web servers, endpoints, and other devices.

3.1.8 Microservices

There are services, as we have just seen; and then there are *tiny* services? so tiny that they are called micro-services?

We quote from [9]:

What are Microservices?

Microservices refer to the thousands of independent web standards, programming languages, database platforms, and web server components that are found in the contemporary software development lifecycle as developer tools. From a traditional perspective, enterprise companies once focused on Service-Oriented Architecture (SOA) which represented hardware and software technology that was integrated from a single IT company. With microservices, there are thousands of different components being supported in cloud software applications and web servers from independent development companies or open source communities. IT departments needed a new philosophy to manage microservices in production across isolated multi-tenant environments in hyper-scale public cloud data centers and have widely adopted virtualization solutions with SDDC standards driven by service

mesh technology to solve this. Microservices form the building blocks or fundamental components, platforms, and frameworks on which code is built and operated on web servers in a cloud data center.

Benefits of Microservices

- **Rapid innovation**: Businesses and startups can bring innovation to market more quickly than using monolithic architecture when required to create new functionality for software applications. Customers using web and mobile applications demand new features. Innovative technology receives funding through popular adoption and enterprise uptake. There is an advantage for both IT majors and startups to stay on the cutting edge of programming and development by integrating new microservices.

- **Greater levels of data center automation**: Developers prefer to use certain platforms or standards for their work and this includes support for programming languages and databases in web/mobile applications with microservices. Microservices connect through scripted processes such as APIs that can lead to greater levels of data center automation.

Monolithic architecture versus Microservices architecture

- **Monolithic architecture**: Derived from the IBM mainframe era and Microsoft Windows OS monopoly period of enterprise IT traditionally.

- **Microservices**: Originally evolved from open source communities, third-party developers, and start-ups with independent programmers contributing code that would extend the base functionality of the most popular web server platforms in use. Now, most of the major IT companies release their own microservices and contributions to open source projects where standards are adopted across different verticals and teams from a wider marketplace of solutions on unique fundamentals. Microservices operate on the same principle of developer innovation through open source code solutions for cloud applications, although proprietary-licensed microservices are also common today.

How do Microservices work?

Microservices work primarily by increasing the functionality available on a web server over that which is provided by the default operating system, network, or data center management platform. Some microservices are web server stack extensions like programming language platforms and database frameworks. In order to develop PHP and MySQL applications, the web server environment must be configured with LAMP platform support. Microservice-based solutions compete largely with the Service-Oriented Architecture supplied by Microsoft, Oracle, IBM, and other IT majors in closed-source distributions. Enterprise companies must now support multiple applications written in PHP, Python, Ruby-on-Rails, Java, C++, ASP.net, etc. in tandem or simultaneously in production. In other situations, a data center may have multiple databases frameworks in operation via VMs on multi-tenant hardware. A web server must be configured for support of custom extensions for streaming media, API integrations, or through adding proprietary utilities for analytics. Whenever all these third-party and open source services are added together, they equal thousands of microservices on a typical web server in production, which are then multiplied by millions of VMs at hyper-scale in a public cloud service environment.

History of Microservices

The use of microservices as a term primarily developed after the Web 2.0 era as corporate IT needed a new way to conceptualize the software development environment to reflect the current ecosystem and best practices. The use of microservices for the development of web and mobile applications has led to increased complexity for data center administrators

to support in enterprise operations. Instead of simply contracting with Oracle, IBM, or Microsoft for all required software and data center solutions, companies now must navigate a landscape with millions of open source code projects which offer enterprise-grade solutions for business use. Many web standards have competing versions from different companies. Many programming languages and databases cover the same functionality in use. Each development team has their own preferences for tools and programming methods. Managing all of this complexity in the cloud era within a unified corporate IT department has led to the need to support microservices in professional software development in many complex organizations worldwide.

Understanding Microservices architecture

The best way to understand microservice architecture is to picture the layers of software in a web server stack. The operating system can be either Windows, Linux, or BSD for the web server. There are tools for data center management and load balancing on the network. There is a choice of Apache, IIS, NGINX, Caddy, Tomcat, etc. for the web server. Next is the layer for the installed programming language support, such as PHP, ASP.net, Python, Ruby, Perl, Java, and Go. Following is the layer for database frameworks like MySQL, MSSQL, PostgreSQL, and MongoDB. Another layer for caching utilities like Varnish, Redis, CDNs, and optimization utilities. Other layers of support include edge servers, serverless platforms, and AI/ML integration. Within a public cloud ecosystem, there are thousands of microservices operating simultaneously which need to be managed in the service mesh to support interoperability, routing, and communications.

Service-Oriented Architecture (SOA) vs. Microservices

Service-Oriented Architecture (SOA) usually comes from a single vendor or includes a package of hardware and software solutions that can be deployed for a well-defined industry need. SOA supports the data center, the web server, and the stack layer through different distributions or product models. SOA is available in both proprietary and open source solutions by vendor. Networking equipment and fiber connections are managed in different ways in cloud data centers vs. in-house private data centers. Microservices are required for innovation in the current state of the software development landscape. Even the largest corporations need to adopt microservices in support of Agile programming teams bringing new web/mobile applications to market, where product support can include thousands of brands or domains. Microservices are frequently required for the modernization and containerization of legacy enterprise software.

Example 3.4. **Microservices by you.** Consider the ATM application we presented in Example 2.17, on page 32.

1. Write pseudocode for this application; be as detailed as you want.
2. Discuss how a monolithic implementation would work.
3. Suppose that this implementation can support a load of x requests per second. Discuss what part(s) of the pseudocode could be the bottleneck(s).
4. What could you do to accommodate a load of $2x$ requests per second?
5. Can you suggest a split of this application into microservices? if yes, how many? why did you choose such a split?
6. Could you have come up with a different split? Can you compare pros and cons between the two?

We will revisit this problem in an actual lab setting, once we learn about how Kubernetes scales applications up (and down) in Section 5.4.6, page 126. The advantages of microservices will then become clearer. △

Example 3.5. **Microservices by big players.** See [10] for an interesting discussion about the use of microservices in Amazon, Netflix, Uber, and Etsy. △

3.1.9 Feedback-based control

3.1.9.1 Feedback is everywhere

Like the notion of Service, the notion of feedback is everywhere. In everyday life, feedback is used to "... *supply information* about reactions to a product, a person's performance of a task, etc. which is used as a basis for *improvement*".

In engineering, control systems are designed to achieve desired objectives (improvement). A Feedback-based control system takes into account information regarding the objective and adjusts action in an effort to achieve it. Feedback-based control is ubiquitous in engineering systems, small- or large-scale ones. Here is a couple of examples.

Example 3.6. **Annoying feedback.** Typically, the signal captured by a microphone is amplified before the amplified output is sent to a speaker. If the microphone is close to the speaker, it captures its output, amplifies it again and sends it to the speaker. This creates the annoying, screeching[2], high-pitched "whine" or "howl" sound. (Well, not so annoying if it comes from a punk rock band, according to some.) △

Example 3.7. **Feedback in transistor amplifiers.** See [29] for a great discussion and explanation with figures, if you can understand circuits, that is. △

Example 3.8. **OSPF routing protocol in an enterprise network; STP protocol in a Local Area Network.** The objective of the protocol is to keep all hosts connected, despite link failures. The feedback is a link failure. The control action is to recalculate routes to maintain connectivity. △

Example 3.9. **Internet of Things, the largest system of all.** IoT, the up-and-coming Internet of Things, is the "largest" engineering system in which feedback plays a central role. Sensors "measure" the state of a system and feed their output to a processing facility. This facility could be the cloud itself; Amazon, for example, offers a variety of IoT services to facilitate IoT usecases, as described in [23]. The result of this processing then feeds actuators in an effort to achieve the system's objectives. Figure 3.1 applies here as well. △

Example 3.10. **The plethora of Self-??? systems.** All such systems use some form of feedback to perform their functions. Examples are:

1. Self-healing: Automatic discovery, and correction of faults;
2. Self-optimizing: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements
3. Self-regulating: A system that operates to maintain some parameter, e.g., Quality of service, within a reset range without external control;
4. Self-learning: Systems use machine learning techniques such as unsupervised learning which does not require external control;

△

[2] `https://www.youtube.com/watch?v=SEy_7dAFlqk`

3.1.9.2 Feedback in the cloud computing world

We will discuss feedback in great detail in two specific areas: Server load balancing in Section 6.7 and Kubernetes in Section 5.4.6.

MAPE (Monitor, Analyze, Plan and Execute) or MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) is a generic framework for feedback-based design in the IT world. It was introduced by IBM in 2005 [12]. The basic idea is described in Figure 3.1.

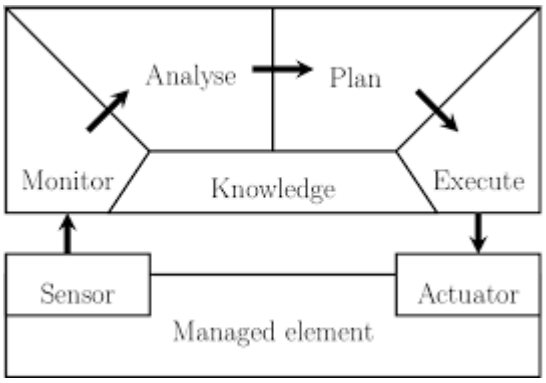


Fig. 3.1: Feedback in the MAPE framework.

3.1.9.3 What's challenging in feedback-based systems?

We'll address this question in detail in Section 5.4.6.

3.2 Cloud characteristics

We have already presented five essential characteristics of cloud computing in Section 1.1.6. It is worthwhile to revisit each one with an analysis perspective in mind, in order to gain insight for the purposes of architecting. Let's take the *Rapid elasticity* characteristic, for example:

Rapid elasticity. *Capabilities can be rapidly and elastically provisioned, in some cases automatically, to scale rapidly outward and inward commensurate with demand.* To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

If we dissect the above paragraph carefully, a number of analysis questions arise:

- What exactly is a capability?
- How many capabilities are there?
- How do we measure “rapidly”?
- Why both outward and inward scaling?
- Under what conditions is “commensurate with demand” scaling possible?

A good handle in such analysis questions is imperative for a great design, would you not agree?

3.3 Cloud taxonomy

Two criteria have been used for providing a taxonomy of cloud environments. Note that the taxonomies can be mixed (i.e., any *service* can be offered in any *deployment*).

You may ask a silly question again: What do we gain by studying the two taxonomies? Let’s assume that you ask this question wearing your architect hat. Having a taxonomy in mind helps you see where in the bigger picture lies the specific problem you are called to solve, as a specialist wearing one of the several hats we mention in Section 6.1. Constraints, tradeoffs, building blocks are a lot clearer with a taxonomy in mind. There is an added bonus too: a taxonomy makes adapting a solution to a different cloud environment a lot easier...

3.3.1 Cloud deployment models

The first criterion had to do with ownership of the environment and the fee structure agreed upon by the provider and consumer actors.

We quote from [1] (the examples are ours):

> There are 4 main types of deploying a cloud: private clouds, public clouds, hybrid clouds, and multiclouds.
>
> **1. Public clouds** Public clouds are cloud environments typically created from IT infrastructure not owned by the end user. Some of the largest public cloud providers include Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure.
>
> Traditional public clouds always ran off-premises, but today’s public cloud providers have started offering cloud services on clients’ on-premise data centers. This has made location and ownership distinctions obsolete.
>
> All clouds become public clouds when the environments are partitioned and redistributed to multiple tenants. Fee structures aren’t necessary characteristics of public clouds anymore, since some cloud providers (like the Massachusettes Open Cloud) allow tenants to use their clouds for free. The bare-metal IT infrastructure used by public cloud providers can also be abstracted and sold as IaaS, or it can be developed into a cloud platform sold as PaaS.
>
> **2. Private clouds** Private clouds are loosely defined as cloud environments solely dedicated to a single end user or group, where the environment usually runs behind that user or

group's firewall. All clouds become private clouds when the underlying IT infrastructure is dedicated to a single customer with completely isolated access.

But private clouds no longer have to be sourced from on-prem IT infrastructure. Organizations are now building private clouds on rented, vendor-owned data centers located off-premises, which makes any location and ownership rules obsolete. This has also led to a number of private cloud subtypes, including:

Example 3.11. **Facebook.** Facebook is arguably the most known example of a private cloud. According to `https://datacenters.fb.com/`, it is housed on 18 data centers located in the United States, Europe and Singapore. All Facebook apps (e.g., Facebook, Oculus, Instagram, Messenger, and WhatsApp) are hosted in these datacenters. Figure 3.2 depicts one of the newest datacenters located in Los Lunas, New Mexico.

Fig. 3.2: The Facebook datacenter in New Mexico.

△

2.1 Managed private clouds Customers create and use a private cloud that's deployed, configured, and managed by a third-party vendor. Managed private clouds are a cloud delivery option that helps enterprises with understaffed or underskilled IT teams provide better private cloud services and infrastructure.

2.2 Dedicated clouds A cloud within another cloud. You can have a dedicated cloud on a public cloud (e.g. Red Hat OpenShift Dedicated) or on a private cloud. For example, an accounting department could have its own dedicated cloud within the organization's private cloud.

3. Hybrid clouds A hybrid cloud is a seemingly single IT environment created from multiple environments connected through local area networks (LANs), wide area networks (WANs), virtual private networks (VPNs), and/or APIs. The characteristics of hybrid clouds are complex and the requirements can differ, depending on whom you ask. For example, a hybrid cloud may need to include:

- At least 1 private cloud and at least 1 public cloud
- 2 or more private clouds
- 2 or more public clouds
- A bare-metal or virtual environment connected to at least 1 public cloud or private cloud

But every IT system becomes a hybrid cloud when apps can move in and out of multiple separate—yet connected—environments. At least a few of those environments need to be

sourced from consolidated IT resources that can scale on demand. And all those environments need to be managed as a single environment using an integrated management and orchestration platform.

4. Multiclouds Multiclouds are a cloud approach made up of more than 1 cloud service, from more than 1 cloud vendor—public or private. All hybrid clouds are multiclouds, but not all multiclouds are hybrid clouds. Multiclouds become hybrid clouds when multiple clouds are connected by some form of integration or orchestration.

A multicloud environment might exist on purpose (to better control sensitive data or as redundant storage space for improved disaster recovery) or by accident (usually the result of shadow IT). Either way, having multiple clouds is becoming more common across enterprises that seek to improve security and performance through an expanded portfolio of environments.

3.3.2 Cloud service models

The second taxonomy criterion examines how the services offered by the provider are exposed to the consumer. It basically reflects the level of management the consumer has to exert on the environment.

There are four main types of clouds, in that regard: IaaS, PaaS, SaaS and FaaS. We describe each one separately, in Sections 3.5 through 3.8.

Note that there are a lot more than four "something as a Service" models out there, inside or outside the cloud computing environment. For example, we have seen offerings of Network as a Service, Network management as a Service, Kubernetes as a Service, Disaster Recovery as a Service, datacenter as a Service...

What are the (main) challenges in offering services? This is a question we'll address in greater detail in Chapters 6 and 5. The challenges confront all hats, of course, for both providers as well as consumers. As expected, challenges are greater for the hats who work for a provider.

3.4 Services in public clouds

3.4.1 The "big 3" public clouds (AWS, Azure, GCP)

3.4.1.1 AWS

We quote from [18]:

AWS serves over a million active customers in more than 240 countries and territories. We are steadily expanding global infrastructure to help our customers achieve lower latency and higher throughput, and to ensure that their data resides only in the AWS Region they specify. As our customers grow their businesses, AWS will continue to provide infrastructure that meets their global requirements.

The AWS Cloud infrastructure is built around AWS **Regions** and **Availability Zones**. *An AWS Region is a physical location in the world where we have multiple Availability Zones.*

Availability Zones consist of one or more discrete data centers, each with redundant power, networking, and connectivity, housed in separate facilities. These Availability Zones offer you the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than would be possible from a single data center. The AWS Cloud operates in 80 Availability Zones within 25 geographic Regions around the world, with announced plans for more Availability Zones and Regions. For more information on the AWS Cloud Availability Zones and AWS Regions, see AWS Global Infrastructure.

Each Amazon Region is designed to be completely isolated from the other Amazon Regions. This achieves the greatest possible fault tolerance and stability. Each Availability Zone is isolated, but the Availability Zones in a Region are connected through low-latency links. AWS provides you with the flexibility to place instances and store data within multiple geographic regions as well as across multiple Availability Zones within each AWS Region. Each Availability Zone is designed as an independent failure zone. This means that Availability Zones are physically separated within a typical metropolitan region and are located in lower risk flood plains (specific flood zone categorization varies by AWS Region). In addition to discrete uninterruptible power supply (UPS) and onsite backup generation facilities, data centers located in different Availability Zones are designed to be supplied by independent substations to reduce the risk of an event on the power grid impacting more than one Availability Zone. Availability Zones are all redundantly connected to multiple tier-1 transit providers.

Fig. 3.3: AWS datacenters.

3.4.1.2 Azure

We quote from `https://azure.microsoft.com/en-us/global-infrastructure/` and `https://azure.microsoft.com/en-us/global-infrastructure/global-network/#overview`.

Microsoft's Azure is housed on 200+ physical datacenters, arranged into 60+ regions, more than any other cloud provider.

Some notable features:

- Tenants can connect at upto 100 Gbps;
- 165,000 miles of lit fiber optic and undersea cable systems
- More than 185 global network POPs
- IP traffic stays entirely within our global network and never enters the public Internet
- Ultra-low latency in Los Angeles, Miami, Vancouver metros

We quote from [28]: Consumers can connect to an Azure datacenter over a private connection. Figure 3.4 shows how.

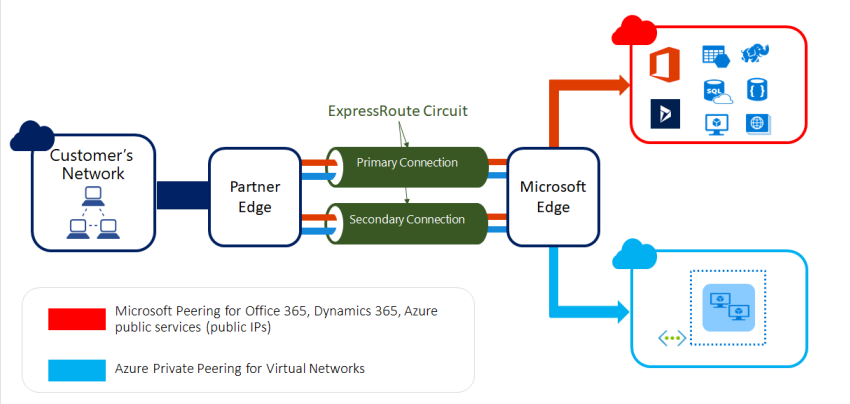


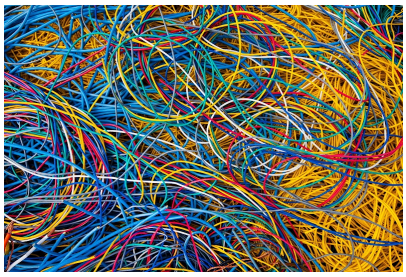
Fig. 3.4: Azure expressroute connectivity; see [33] for details.

3.4.1.3 GCP

We got the following information from [30]:

> We'll mention the technical benefits of architecting in the next two sections. Here we'll briefly mention a "side effect": architecture destroys art. Figure 3.6 depicts cabling inside a typical GCP datacenter before and after one of the architectural principles we outline in Section 4.5.2 was applied (can you tell which one?), in the quest of meeting objectives (like the one described in Section 4.2.2).
>
> Read [32] for information on how Google organizes their datacenters into Regions, Availability Zones and Clusters.

Fig. 3.5: GCP datacenters.



(a) It was artistic before...

(b) and look at it now...

Fig. 3.6: The effect of architecting: destroys art?

3.4.2 Basic services in public clouds

Cloud providers introduce new services as we speak. The top 3 offer over two hundred such services each. The services listed below are among the “basic ones”:

1. Authentication (Identity and Access Management)
2. Connectivity to customer premises (public internet, Private WAN)
3. Discovery (DNS)
4. Compute
5. Storage (block, object etc.)
6. Analytics

We will describe such services in detail in due time.

3.4.3 All services in AWS, Azure, GCP

References [13], [14] and [15] provide a full list of AWS, Azure services respectively. The lists have been updated in December 2021.

Example 3.12. **Overview of Amazon Web Services.** Reference [18] provides a nicely grouped list of all Amazon services (updated on August 5, 2021). △

Example 3.13. **Comparison of services in all 3.** Reference [16], produced by Google and last updated on August 31, 2021, lists generally available Google Cloud services and maps them to similar offerings in Amazon Web Services (AWS) and Microsoft Azure. △

3.5 Infrastructure as a Service (IaaS)

3.5.1 What is IaaS?

The NIST definition [2] of cloud computing defines Infrastructure as a Service as:

> The capability provided to the consumer is to **provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications**.

The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Figure 3.7 (reprinted from [1]) emphasizes the point made in the paragraph above regarding who shoulders the burden of administering the infrastructure.

We will discuss tools that are applicable to this type of service in Chapter 5.

Example 3.14. **IaaS providers.** Pretty much every provider offers IaaS. We (randomly) mention one: it's worth browsing `https://www.inmotionhosting.com/cloud/flex-metal-iaas`.

See [19] for 16 more, if interested. If you want a top 10 list, here is one (of many) [34]. △

3.5.1.1 What's the (main) problem IaaS solves for the cloud consumer?

The question is rather silly because it's vague: the consumer may be a single user (e.g., you as a student); may be an organization that has users, IT personnel and application developers.

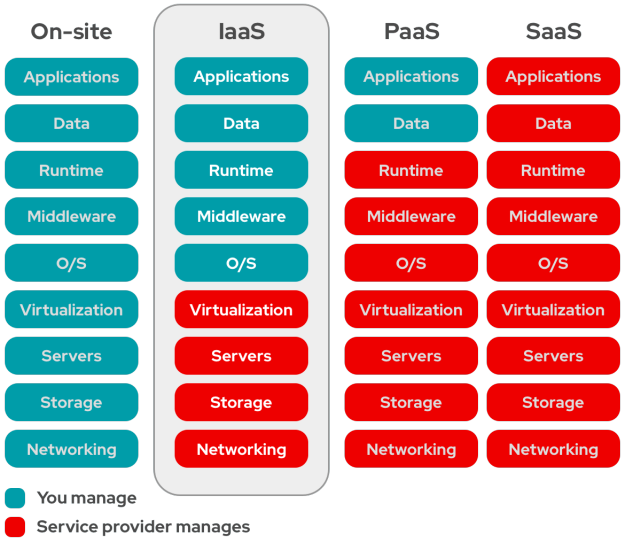


Fig. 3.7: Infrastructure-as-a-service (IaaS).

IaaS targets the IT personnel in a consumer's organization. We'll have to go with the *time to deploy infrastructure*. Anecdotal quotes mention reductions from months down to minutes.

If we had to mention other candidates in a list of "top IaaS advantages", we can quote the following list from[3] `https://www.comptia.org/content/articles/what-is-iaas`:

- Pay for What You Use: Fees are computed via usage-based metrics
- Reduce Capital Expenditures: IaaS is typically a monthly operational expense
- Dynamically Scale: Rapidly add capacity in peak times and scale down as needed
- Increase Security: IaaS providers invest heavily in security technology and expertise
- Future-Proof: Access to state-of-the-art data center, hardware and operating systems
- Self-Service Provisioning: Access via simple internet connection
- Reallocate IT Resources: Free up IT staff for higher value projects
- Reduce Downtime: IaaS enables instant recovery from outages
- Boost Speed: Developers can begin projects once IaaS machines are provisioned
- Enable Innovation: Add new capabilities and leverage APIs
- Level the Playing Field: SMBs can compete with much larger firms

[3] On a closer look, a lot of these are not unique to IaaS.

3.6 Platform as a Service (PaaS)

3.6.1 What is PaaS?

The NIST definition [2] of cloud computing defines Platform as a Service as:

> The capability provided to the consumer is to **deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider**.

The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Figure 3.8 (reprinted from [1]) emphasizes again the reduced burden of administering the infrastructure.

We will discuss tools that are applicable to this type of service in Chapter 5.

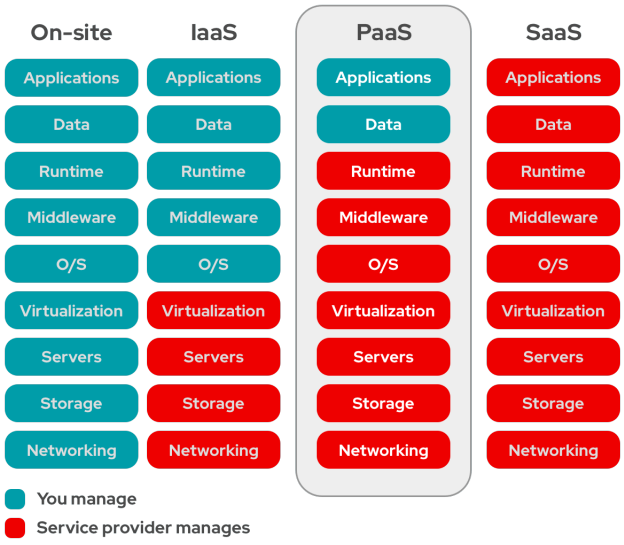


Fig. 3.8: Platform-as-a-service (PaaS).

Example 3.15. **A database PaaS provider.** Pretty much every provider offers PaaS. There are some that focus on specific platforms and excel in them. We (randomly)

mention Snowflake[4], a provider that specializes in databases: check them out at `https://www.snowflake.com/`. △

Example 3.16. **A list of PaaS providers.** If you're interested in working for a PaaS provider, start your search with [20] for some names. If you want a top 10 list, here is one (of many) [35]. △

3.6.1.1 What's the (main) problem PaaS solves for the cloud consumer?

PaaS targets a consumer's application developers.

With this target in mind, we'll have to go with this: PaaS enables the consumer's software developers to focus only on application development (that's what they are supposed to do best) - no worries to build and maintain the infrastructure associated with running the applications (that's probably uninspiring to them).

Example 3.17. **What if I want to develop an application that runs on several clouds?** Check [24] for details. Nutanix, [5] as Figure 3.9 depicts, supplies a generic platform for programmers to develop applications that can run on *hybrid and/or multicloud* environments.

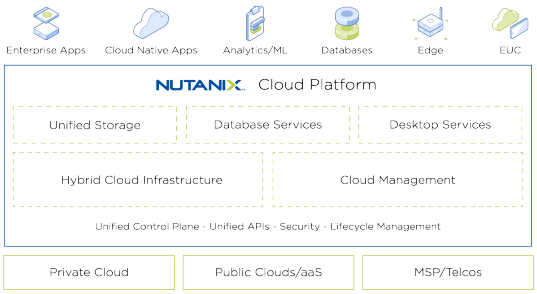


Fig. 3.9: The Nutanix platform.

△

[4] The funny thing is that Snowflake provides its service on top of the big three (IaaS) providers who also provide database services.

[5] a company with presence in Durham and appetite to hire NCSU graduates as opposed to Duke graduates

3.7 Software as a Service (SaaS)

3.7.1 What is SaaS?

The NIST definition [2] of cloud computing defines Software as a Service as:

> The capability provided to the consumer is to **use the provider's applications running on a cloud infrastructure**.

The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Figure 3.10 (reprinted from [1]) emphasizes again the reduced burden of administering the infrastructure.

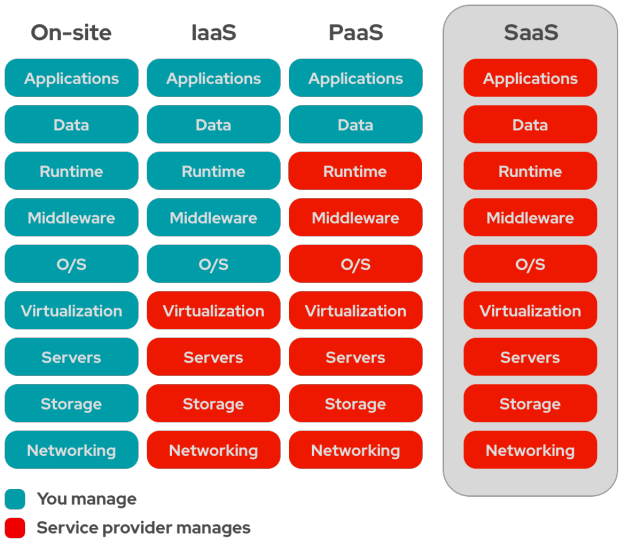


Fig. 3.10: Software-as-a-service (SaaS).

Example 3.18. **A list of SaaS providers.** If you're interested in working for a SaaS provider, start your search with [21] for some names. Unlike the lists for IaaS and PaaS, the SaaS top list in [36] discusses 30 of them. Any idea why? △

3.7.1.1 What's the (main) problem SaaS solves for the cloud consumer?

SaaS targets a consumer's users.

With this target in mind, we'll have to go with the *simplicity in using an application*; after all, that's what the users expect, just click like monkeys - no worries to save and backup a latex report you edit, just use overleaf!

3.8 Function as a Service (FaaS)

3.8.1 What is FaaS?

We quote from [31]:

> Function-as-a-Service, or FaaS, is a kind of cloud computing service that **allows developers to build, compute, run, and manage application packages as functions without having to maintain their own infrastructure**.
>
> FaaS is an event-driven execution model that runs in stateless containers and those functions manage server-side logic and state through the use of services from a FaaS provider.

Example 3.19. **FaaS providers.** All major providers offer FaaS, even though the exact list of functions may differ. Names on the list include:

- AWS: offers Lambda functions
- Azure: offers Azure functions
- CGP: offers Google Cloud Functions
- IBM: offers Cloud Functions

There is also OpenFaaS, `https://www.openfaas.com/`, an open source offering. △

3.8.1.1 What's the (main) problem FaaS solves for the cloud consumer?

Like SaaS, FaaS targets the user in the consumer organization.

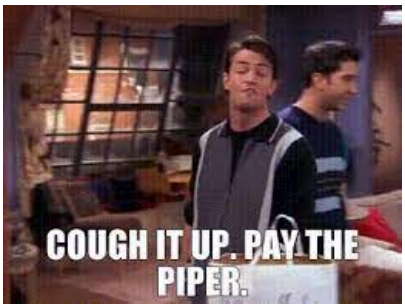
We'll have to go with the *simplicity in using an application*.

3.9 Time to pay the piper: cost of a Service

They say there is no free lunch in engineering; at the end of the day, consumers pay providers for their services.

We quote from [22]:

(a) Piper 1, IaaS.



(b) Piper 2, SaaS.

Fig. 3.11: Pipers...

Pay-as-you-use (or pay-per-use) is a payment model in cloud computing that charges based on resource usage. The practice is similar to the utility bills (e.g. electricity), where only actually consumed resources are charged.

One major benefit of the pay-as-you-use method is that there are no wasted resources (that were reserved, but not consumed), which can be a source of significant losses for the companies.[1] Users only pay for utilized capacities, rather than provisioning a chunk of resources that may or may not be used.

Payment model concept evolution Cost efficiency is one of the most distinctive and advertised benefits of cloud computing alongside the ease of use.[2] Due to cloud computing rapid development, the utilized payment model is also evolving.

Subscription is the most basic payment model that provides periodic access to a product or service. The main benefit is a predictable fixed cost, which is independent of the consumption rate or whether the service is used at all. The downside of the model is that it is difficult to forecast the consumption beforehand, which leads to overallocation.[3]

Pay-as-you-go (also may be referred to as pay-as-you-run, pay-as-you-allocate, etc.) is the most frequently used payment model at the moment. The main idea is that users only pay for the provisioned server (virtual machine) when it is running (going). However, in terms of resources pay-as-you-go approach charges fully for the allocated resources (i.e. VM limit) regardless of the actual consumption.

Pay-as-you-use is the most recent payment model in cloud computing that emerged after integration and popularization of the containers in the clouds. It is centered on the containers' ability to dynamically scale the amount of provided resources without downtime (vertical scaling). As a result, the charges can be made based on the actual consumption during the specific time.[4]

Role in solving the right-sizing problem *Right-sizing* is a process of reserving the cloud computing instances (containers, VMs, or bare metal) with enough resources (RAM, CPU, storage, network) to achieve a sufficient performance at the lowest cost possible.[5]

Right-sizing aims to solve two problems in cloud computing:

• *Overallocation*, which leads to inefficient utilization of the cloud infrastructure and overpayment for resources that are not actually used.

• *Underallocation*, which results in resource shortage that causes performance issues or even downtime of the hosted projects, leading to the poor end-user experience, missed clients, and revenue losses.

Payment Model	Charged when the server is stopped?	Charged for unused resources?	Billing Period	Summary
Subscription	+	+	long (month or year)	Charged full price at once. Similar to purchasing a server
Pay-as-you-go	-	+	short (second or hour)	Charged for the full capacity when the server is running. Similar to renting a server.
Pay-as-you-stay	-	+	short (minutes or hour)	Charged for the consumed hours. Similar to renting a car.
Pay-as-you-use	-	-	short (second or hour)	Charged for the consumed resources. Similar to renting resources.

Fig. 3.12: Payment models.

Currently, the pay-per-use model is the most efficient answer to the right-sizing problem. It allows avoiding manual prediction on the required server size by shifting this responsibility to the precise tools offered by modern cloud hosting providers. As a result, applications are automatically provided with the exact amount of resources to serve the on-going load.

Discrepancy in Terminology Due to lack of unified terminology and relative novelty, the pay-as-you-use term in cloud computing is often confused with similar ones like pay-as-you-go, pay-as-you-run, pay-as-you-allocate, etc. The terms are mixed up especially frequently in the marketing materials due the appealing word structure ”pay only for the resources you use”, while with pay-as-you-go pricing is primarily based on the instance size but not real resource consumption.

References

1. Becky Peterson (December 1, 2017). ”Companies waste $62 billion on the cloud by paying for capacity they don’t need, according to a report”. Business Insider. Retrieved 11 April 2021.
2. Linda Rosencrance (November 16, 2020). ”Breaking Down the Cost of Cloud Computing”. Whatis.com. TechTarget. Retrieved 11 April 2021.
3. Meghan Liese (October 21, 2020). ”Is Cloud Waste Inevitable as Companies Move to the Cloud?”. The New Stack. Retrieved 11 April 2021.
4. Ruslan Synytsky (March 28, 2018). ”Deceptive Cloud Efficiency: Do You Really Pay As You Use?”. Forbes. Retrieved 11 April 2021.
5. Ruslan Synytsky (November 5, 2020). ”The Right-Sizing Problem in Cloud Computing, and How to Solve It”. the New Stack. Retrieved 11 April 2021.

3.9.1 Pricing services at the big 3

Amazon's pricing is described in `https://aws.amazon.com/pricing/`; Azure's in `https://azure.microsoft.com/en-us/pricing/` and GCP's in `https://cloud.google.com/pricing`.

It's not a simple task for a consumer to figure the actual cost of a cloud solution that employs several services. There are tools for that and even a financial architect role!

3.10 Problem Section

Problems on fundamental notions in analysis of a cloud

Problem 3.1. Who's most important? Consider the technologies of virtualization and service. The first "works in the shadows" (inside servers, storage devices, network equipment that make the cloud computing possible). The second is the "face" of cloud computing (what the cloud consumer cares and sees).

1. Which technology is more important, in your opinion? Justify your answer.

Problem 3.2. The abstract concept of service. Consider the bus service at NCSU.

1. Describe the service in your own words.
2. Who is the provider?
3. Who is the consumer?
4. What are the benefits to the provider?
5. What are the benefits to the consumer?
6. How does the service get discovered by the consumer?
7. How does the provider "advertise" the service?

Problem 3.3. The abstract concept of service. Consider the internet, IP, its layer 3 protocol, and a layer 4 protocol, say TCP.

In protocol parlance, the layer 3 protocol supplies a service to the layer 4 protocol.

1. Describe the service in your own words.
2. What are the benefits to the consumer?

Problem 3.4. The abstract concept of service. Consider the internet, a layer 5 protocol, say https, and two layer 4 protocols, namely TCP and UDP.

In protocol parlance, the layer 4 protocol supplies a service to a layer 5 protocol. In our parlance, TCP and UDP are providers and any layer 5 protocol is a consumer. A protocol has a transmit (X) and a receive (R) component. (https calls them client and server respectively.) So, we have a producer/receiver pair at X and another pair at R.

1. Find out which layer 4 protocol https is using.
2. Describe, in your own words, the service this protocol provides to https.
3. How are the producer and consumer implemented in a computing device?
4. In your opinion, is it possible to have a provider serve several consumers in that device? Explain with an example.
5. In your opinion, is it possible to have the https consumer be served by several providers in that device? Explain with an example.
6. How does the producer know the consumer in that device?
7. How does the service get discovered by the consumer in that device?
8. How does the provider "advertise" the service in that device?

Problem 3.5. The concept of microservice. Consider the banking application of servicing ATM transactions.

1. Describe the logic that this application has to implement.
2. Define a set of microservices that can be used to implement this logic.
3. Can you split one of these microservices into at least two "smaller" ones?
4. Define interfaces for each microservice.
5. Supply a diagram that depicts "who talks to whom".

Problem 3.6. Virtualization, networks. Consider the internet, IP, its layer 3 protocol, and two layer 4 protocols, namely TCP and UDP.

The system in the definition of virtualization we gave in Section 3.1.1 is the internet.

1. Explain how TCP virtualizes the internet. More specifically, what is the property TCP gives to the internet, that the internet does not (natively) have?
2. Describe, in any detail you like, what mechanisms TCP uses in order to achieve the property.
3. Who (not which hat) benefits from this virtualization?
4. Which hat benefits from this virtualization?
5. Which hat pays the price?
6. Explain why UDP does not virtualize the internet like TCP does.
7. Search "what protocols use UDP" and "what protocols use TCP". Explain why some protocols don't care to use an internet full of virtues...

Problem 3.7. Virtualization, servers. Consider any one of the compute servers we supplied in the references section of Chapter 2.

1. Define the requirements of a VM.
2. How many VMs can a specific server support?

Problem 3.8. Feedback-based control. Give an example from your personal, college life in which feedback was used to achieve an objective. Describe the problem using the terminology of the MAPE framework we presented in Figure 3.1, page 57. More specifically:

1. State the objective clearly.
2. What is the managed element?
3. What is the sensor and actuator?
4. What analysis is done? Where? Who did the analysis?
5. How the results of the analysis were communicated to the actuator?
6. Discuss the benefits, if any.
7. Discuss the challenges, if any.

Problems on cloud models

Problem 3.9. ?aaS. The following are offerings of services. The question is what kind? Justify your answer. (Google the name, if you are not familiar with the company that offers the service.)

1. Overleaf?
2. Dropbox?
3. Google docs?
4. Gmail?
5. Netflix?

Problem 3.10. SaaS. In Section 3.7.1.1, we have presented the (main) problem SaaS solves for the cloud consumer.

We can also take the cloud provider's point of view. What in your opinion, is an advantage for a SaaS provider?

Problem 3.11. SaaS. Give two examples of SaaS that we have not discussed in these notes or the lectures.

Problem 3.12. PaaS. Give two examples of PaaS that we have not discussed in these notes or the lectures.

Problem 3.13. IaaS. Give two examples of IaaS that we have not discussed in these notes or the lectures.

Problem 3.14. FaaS and serverless computing. In Section 3.8.1, we briefly presented FaaS, the newest kid on the block among the four main service models.

FaaS is a way to implement *serverless computing*.

1. Describe what serverless computing is.
2. Mention its advantages for the application developer.
3. Describe how FaaS implements serverless computing.

Problems on specific services

Problem 3.15. DNS service. Search the web for information on Route 53, Amazon's cloud DNS service or use [25].

Summarize, in your own words, the DNS service offered.

Problem 3.16. DNS service. Search the web for information on Google's cloud DNS service or use [26]. Google also offers a public, free DNS service; see [27]. Reportedly, the public service handles over a trillion queries per day.

1. Summarize, in your own words, the DNS service offered.
2. Why, in your opinion, Google offers separate DNS services?

Problems on the big three cloud providers

Problem 3.17. Tenants in AWS. According to AWS, they serve about a million tenants in their datacenters. On average, how many tenants per datacenter? Per server?

Problem 3.18. Pricing. The big three cloud providers offer several types of compute service; however, they may not use the same name or exactly the same features for them.

1. Find a compute service that has (almost) the same features in all three.
2. Describe the features. Outline the differences, if any.
3. Find out, if possible, the cost of the service (i.e., up-front, per hour charges).

Problem 3.19. Burstable compute instances. Search the AWS documentation about burstable compute instances.

1. Describe the key concepts of this service.
2. State succinctly what is the benefit for the consumer.
3. State succinctly what is the benefit for the provider.

Problem 3.20. Service pricing. Find out the cost per hour for using any type of compute instance in any cloud provider.

1. Find out the cost of buying a (desktop, laptop) computer with similar features.
2. Calculate when a consumer will break even. State your assumptions clearly.

Problem 3.21. Azure features. Find out about Azure's ExpressRoute. Microsoft promises "ultra-low latency in Los Angeles, Miami, Vancouver metros".

1. What can they possibly mean by that?
2. In your opinion, how could they achieve it?

Huh?

Problem 3.22. Virtualization. Virtualization is everywhere, we said in Section 3.1.1. But, like beauty, "everywhere" is in the eye of the beholder.

1. Find examples of virtue in medieval visual arts.
2. Find examples of virtue in 20th century visual arts.
3. In your opinion, do you see any difference? Has virtualization in the visual arts improved, in any sense?
4. Any virtue in rock and roll music?
5. A quick search on `www.lyrics.com` on January 3, 2022, unearthed 4,101 lyrics, 16 artists, and 10 albums matching virtue. Write down the lyrics of one such song.
6. Do you find any resemblance between the virtue in that song and, say, the virtue in a VLAN? or a VM?

References

References marked as Handout H3.x are required reading material; you will find them in the required handouts binder in moodle. The rest are used in some form in the text or problems.

1. **Handout H3.1:** Cloud types, https://www.redhat.com/en/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud
2. **Handout H3.2:** The NIST Definition of Cloud Computing, Special Publication 800-145, (same as Handout H1.1) https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
3. **Reference R3.1:** What is Server Virtualization? https://www.vmware.com/topics/glossary/content/server-virtualization.html
4. **Reference R3.2:** Virtualization basics, Author Notes, 2020.
5. **Reference R3.3:** What is a virtual machine? https://www.vmware.com/topics/glossary/content/virtual-machine.html
6. **Reference R3.4:** What is a container? https://www.vmware.com/topics/glossary/content/containers.html
7. **Reference R3.5:** VMs vs containers, https://www.vmware.com/topics/glossary/content/vms-vs-containers.html
8. **Reference R3.6:** What is a service mesh? https://www.vmware.com/topics/glossary/content/service-mesh.html
9. **Reference R3.7:** What is a microservice? https://www.vmware.com/topics/glossary/content/microservices.html
10. **Reference R3.8:** 4 Microservices Examples: Amazon, Netflix, Uber, and Etsy, https://blog.dreamfactory.com/microservices-examples/
11. **Reference R3.9:** Joe the IT Guy, "ITSM Basics: What Is a Service?", February 2018, https://www.joetheitguy.com/itsm-basics-what-is-a-service/
12. **Reference R3.10:** An architectural blueprint for autonomic computing, June 2005, https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
13. **Reference R3.11:** The full list of AWS services https://www.eckher.com/c/21gjdl7gz4
14. **Reference R3.12:** The full list of Azure services https://www.eckher.com/c/21h7dv3g91
15. **Reference R3.13:** The full list of Google Cloud Platform Services https://cloud.google.com/terms/services
16. **Reference R3.14:** Compare AWS and Azure services to Google Cloud, https://cloud.google.com/free/docs/aws-azure-gcp-service-comparison
17. **Reference R3.15:** SLA for Azure Kubernetes Service (AKS), last updated March 2020, https://azure.microsoft.com/en-us/support/legal/sla/kubernetes-service/v1_1/
18. **Reference R3.16:** Overview of Amazon Web Services, AWS Whitepaper https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf#global-infrastructure
19. **Reference R3.17:** Timothy Shim, 17 Popular Infrastructure (Software) as a Service (IaaS) Examples, Updated: Oct 21, 2021, https://www.webhostingsecretrevealed.net/blog/web-business-ideas/iaas-examples/
20. **Reference R3.18:** Timothy Shim, 15 Popular Platform as a Service (PaaS) Examples, Updated: Dec 22, 2021, https://www.webhostingsecretrevealed.net/blog/web-business-ideas/paas-examples/

21. **Reference R3.19:** Timothy Shim, 17 Popular Software as a Service (SaaS) Examples, Updated: Oct 06, 2021, https://www.webhostingsecretrevealed.net/blog/web-business-ideas/saas-examples/
22. **Reference R3.20:** Pay-as-you-use, https://en.wikipedia.org/wiki/Pay-as-you-use
23. **Reference R3.21:** IoT services in AWS, https://aws.amazon.com/free/iot/
24. **Reference R3.22:** Nutanix - What We Do, https://www.nutanix.com/sg/what-we-do
25. **Reference R3.23:** Amazon Route 53, https://aws.amazon.com/route53/
26. **Reference R3.24:** Google's cloud DNS service, https://cloud.google.com/dns
27. **Reference R3.25:** Google's public DNS service, https://en.wikipedia.org/wiki/Google_Public_DNS
28. **Reference R3.26:** What is Azure ExpressRoute? https://docs.microsoft.com/en-us/azure/expressroute/expressroute-introduction
29. **Reference R3.27:** Feedback in Bipolar Junction Transistors, https://www.allaboutcircuits.com/textbook/semiconductors/chpt-4/feedback/
30. **Reference R3.28:** Google Cloud Platform locations, https://cloud.google.com/about/locations
31. **Reference R3.29:** What is Function-as-a-Service (FaaS)? https://www.redhat.com/en/topics/cloud-native-apps/what-is-faas
32. **Reference R3.30:** GCP Regions and zones, https://cloud.google.com/compute/docs/regions-zones
33. **Reference R3.31:** ExpressRoute partners and peering locations, https://docs.microsoft.com/en-us/azure/expressroute/expressroute-locations-providers
34. **Reference R3.32:** Top 10 IaaS Providers, https://blog.back4app.com/iaas-providers/
35. **Reference R3.33:** Top 10 PaaS Providers, https://blog.back4app.com/Paas-providers/
36. **Reference R3.34:** Top 30 SaaS Providers, https://www.cledara.com/blog/top-300-saas-companies

Topic 4
Architecture requirements, process, principles and best practices

Objectives: After reading this chapter, you should be able to: (a) describe the business requirements for which you are architecting, (b) list the building blocks at the disposal of a cloud architect, (c) describe the basic design principles, (d) describe some of the best practices, and, (e) highlight the commonalities of the AWS and GCP architecture frameworks.

The purpose of any architecture is to suggest ways to satisfy (generically-stated) business requirements. In this chapter, we first list such requirements of interest to two of the actors in the ecosystems, namely cloud consumers and providers. We then suggest a process for creating an architecture for the problem at hand. Such a process can be adhoc or use tried and true principles. Finally, we conclude the chapter with a list of best practices: such practices are good starting points for a novice architect...

4.1 The generic business requirements of a cloud consumer

What is a business requirement? Loosely speaking,

a (cloud consumer's) business requirement **states what the cloud consumer wants from a service**.

Note that in a consumer organization requirements may be posed by any entity - users, designers, administrators, owners... Most requirements are common to a provider as well; the level of importance may, however, be different in a provider environment.

Upon a first glance, one may observe that business requirements are stated somewhat vaguely. That happens if one forgets that these statements merely state the consumer's wish. *Nothing implies that they are doable*; they *may be conflicting* too. The transformation into technical requirements (that we'll discuss in the next chapter) addresses the do-ability question.

One final comment: requirements must be **observable** - there is no point in coming up with a design otherwise.

4.1.1 What are (some of) these requirements in a nutshell?

Here is the list, for convenience of reference. Note the use of actionable verbs in every requirement[1].

Reduce costs
Accommodate time-varying workloads
Be available 24/7
Performance should be high
Operations should be secure
Avoid "vendor lock-in"

4.1.1.1 Cost efficiency, reduction or "optimization"

Consumers must pay the pipers, as we have seen at the end of the previous chapter. So, they naturally want a "bigger bang for their buck", similar to the phrase "more bounce to the ounce". But these are phrases coined in the 1950s; nowadays, and in the context of cloud computing, we convey the same meaning by saying that the cloud consumer would require a better financial deal from the cloud provider when compared to "computing on premise".

Computing on premise, of course has a lot of components. Here are a few.

Example 4.1. **Cost of licenses:** Software packages (e.g., antivirus software) may have license fees that the cloud consumer must pay per laptop/desktop, per year or once. △

Example 4.2. **Cost of hardware.** This refers to computers, disk storage, the entire IT infrastructure. It is a fixed cost, regardless of whether the server is sitting idle or not. △

[1] It is rather too common in the industrial/marketing literature to express requirements without such verbs; for example, stating the requirement simply as "Performance". Does that indicate that the consumer wants minimal performance? mediocre? high?

Example 4.3. **Cost of maintenance.** This includes annual fees that an IT supplier may charge, as well as the salaries of administrative personnel. △

Example 4.4. **The power of buying in bulk.** A large cloud provider like AWS can buy 50,000 servers at a price much lower than 500 small enterprises buying 100 servers each. Indirectly, a consumer can strike a better deal by moving work to a cloud provider. △

4.1.1.2 Ability to accommodate time-varying workloads

As we have discussed in Section 2.3, page 31 and Problem 2.7, page 35, consumer workloads are fluctuating.

A reasonable requirement then is to accommodate the load, even at its peak. Subject to the first requirement, of course.

Example 4.5. **nfl.com** Football fans visit nfl.com throughout the year. The site is a trove of news, stats, videos, even t-shirts.

During the months of late August to mid February, traffic to the site spikes considerably every Sunday afternoon or Monday night or Thursday night or playoff dates or the SUPERBOWL night (can you see why?) compared to other days of the year. △

4.1.1.3 24/7 availability

Intuitively, 24/7 availability of a system is easy to grasp and describe: the system should be able to process a workload request at any time.

What can make availability less than 24/7? Faults of many, many kinds. Here are three examples that cover the whole spectrum (and hopefully give the architect in you mitigation ideas).

Example 4.6. **The easy case, a single server goes down.** In a datacenter with 100,000 (virtual) servers, one will become nonoperational quite often. If this server was dedicated to nfl.com, the service loses 24/7 availability. What's the obvious remedy? have a backup server on the standby, you say? △

Example 4.7. **The difficult case, a datacenter goes down.** If nfl.com is housed in (several servers, but inside) a single datacenter, the service loses 24/7 availability. What's the obvious remedy? have a backup datacenter on the standby, you say? △

Example 4.8. **The disaster case, an entire set of services goes down.** AWS had an outage incident on December 7, 2021; it affected **all of the 200+ services** AWS offered out of a US East datacenter. AWS provided an explanation in `https://aws.amazon.com/message/12721/`. What's the obvious remedy? have a backup cloud provider on the standby, you say? △

Example 4.9. **The disaster case, an entire set of services goes down.** Facebook had an outage in early October 2021. The Facebook, Instagram, Messenger, Whatsapp, and OculusVR services were all unavailable for six hours. The culprit was a botched configuration change to its routers during a routine maintenance mistake. The result was that **all of Facebook's datacenters** were disconnected from the internet. What's the obvious remedy? have a backup "Facebook" on the standby, you say? △

4.1.1.4 High performance

Performance can be defined in a quantitative or qualitative way.

For a generic service, a common quantitative metric is the throughput of the service measured in "requests per second", for example.

Example 4.10. **Quantitative: High throughput banking applications.** Consider the banking application we described in Example 2.17, page 32. The performance requirement may be expressed as the ability to process 1,000 ATM transactions per second - that's 3.6M transactions an hour.

Here is an anecdotal story that relates to this requirement: in 2006, one of the authors (the older one) was given the following problem from an IT company with a large presence in RTP. Wachovia (now part of Wells Fargo) had enough with the complexity of their ATM network. So, they asked this IT company to take back all the hardware and software (ATM machines, the network, the back end (mainframe) servers) and just give Wachovia an ATM service. Wachovia was willing to pay a fee based on the throughput of the system. △

Example 4.11. **Qualitative: High performance video streaming.** Consider Problem 2.7, page 35. A video stream service from nfl.com may be (subjectively) perceived as high performing if the number of interruptions does not exceed the comfort level of a viewer. △

Which entity in the consumer organization you think posed the two examples above?

4.1.1.5 Secure operations

Who doesn't want each and every operation of theirs to be secure? Including but not limited to no malware, no ransom, anonymity, protection of personal data, hack-free environments, no DDoS, etc. etc.

4.1.1.6 Avoid "vendor lock-in"

This is a requirement posed mostly by architects and administrators - not really users. In a design, having to choose options offered by only one vendor is unnecessarily restrictive.

4.2 The generic business requirements of a cloud provider

It should be obvious that the definition of business requirement applies to the producer of a service as well, with the obvious replacement of a word. The availability, performance, security and workload requirements we described for the consumer are also relevant to the producer.

There are a few that are different. We mention next two of them.

4.2.1 Maximize profit

Pipers want to be paid for all the nice music they are piping. AWS is in business to make a buck, after all.

4.2.2 Reduced management complexity

From the description of the IaaS, PaaS and SaaS offerings in Sections 3.5, 3.6 and 3.7 and the figures therein, it should be obvious that the management tasks on the shoulders of the provider's administration team are enormous to say the least.

This team inherits the tasks offloaded by each and every consumer; as expected, the complexity of the "whole is greater than the sum of the parts". Additional tasks arise from the need to keep consumers isolated form each other. Virtualization complicates the management tasks in situations where tools are not available (more on that later in Chapter 5).

4.3 The 6 pillars of the AWS Well-Architected Framework

Pillar[2] is a term AWS is using to describe generic business requirements that refer to both consumer and provider.

We quote from [1] (note the similarities and differences from the requirements we listed in the previous two sections - is this a problem?):

The pillars of the AWS Well-Architected Framework

Operational Excellence The ability to support development and run workloads effectively, gain insight into their operations, and to continuously improve supporting processes and procedures to deliver business value.

Security The security pillar describes how to take advantage of cloud technologies to protect data, systems, and assets in a way that can improve your security posture.

[2] Up until the end of 2021, there were 5 of them [2]. The sixth one, Sustainability, was added at that time to reflect increased concerns about energy consumption in datacenters.

Reliability The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This includes the ability to operate and test the workload through its total lifecycle. This paper provides in-depth, best practice guidance for implementing reliable workloads on AWS.

Performance Efficiency The ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve.

Cost Optimization The ability to run systems to deliver business value at the lowest price point.

(last but not least) **Sustainability** The ability to continually improve sustainability impacts by reducing energy consumption and increasing efficiency across all components of a workload by maximizing the benefits from the provisioned resources and minimizing the total resources required.

4.4 The process for creating an architecture

4.4.1 The process

In a nutshell, the steps in an architect's job[3] can be summarized as follows:

1. Create a list of all **business** requirements for which an architecture is to be produced.
2. Convert this list into another list of **technical** requirements.
3. Search for **options** to address each technical (and hence business) requirement.
4. Evaluate **tradeoffs** for each option.
5. Select an **option for implementation**.
6. Document the selection in a **design document** and/or **architectural diagrams**.

4.4.2 The building blocks of the architecture

Building blocks refer to the elements an architect has at her/his disposal in solving the problem.

Example 4.12. **WAN network architecture.** A WAN network architect has routers/switches with various functionalities, firewalls, links of various technologies and speeds, protocols, topologies, etc. △

[3] Upon closer scrutiny, one can argue that the job of other hats is quite similar. For example, a software programmer that has to implement some functionality starts with step 2, executes steps 3 through 5. The main difference in step 6 is that s/he produces code, something "real", not paper! Given that in a lot of (small) organizations the same person wears the hat of the architect as well as the implementor, this explain the considerable confusion about the distinction of the roles. Tongue-in-cheek then, if you produce paper, you are an architect; if you produce code, you are an architect+programmer ☺

The process in the previous section is generic enough and applies to any subspecialty that we'll mention in Section 6.1. The building blocks, however, depend heavily on the subspecialty. We will mention them in the next chapter, when we discuss problems specific to a subspecialty.

One block applies to most subspecialties: a **service**.

4.4.3 The deliverables of a cloud architect

The generic deliverables are architectural **documents** and/or **diagrams**. Like the building blocks, they depend heavily on the subspecialty.

Who is the recipient of the architect's deliverables? It is important to realize that there may be more than one. An analogy with an architect who designs houses will help make this important point clear. The team that deals with lighting does not need the same docs/diagrams as the team that deals with external fences.

We stress this point by saying that the architect must provide architectural documents and/or diagrams **tailored to the audience**.

Example 4.13. **Convincing the C-level team.** In the early stages of multi-million dollar projects, at least the CEO, CFO, CTO of an actor organization must be convinced about the business merit (CEO), financial soundness (CFO) and technical feasibility (CTO). The master architect must provide different design documents to each. △

Example 4.14. **Moodle on AWS.** Figure 4.1 is an example of an architectural diagram. It depicts the AWS Reference Architecture for moodle, the website we use for this course (reviewed for technical accuracy on December 22, 2021). Details are provided in [3].

△

4.5 Design principles in cloud architecture

4.5.1 What is a principle?

There is considerable confusion in the open literature. The term "principle" means different things to different authors.

This is our definition:

An architectural principle is a methodology or guidance an architect can use in order to select one among several options to address a requirement.

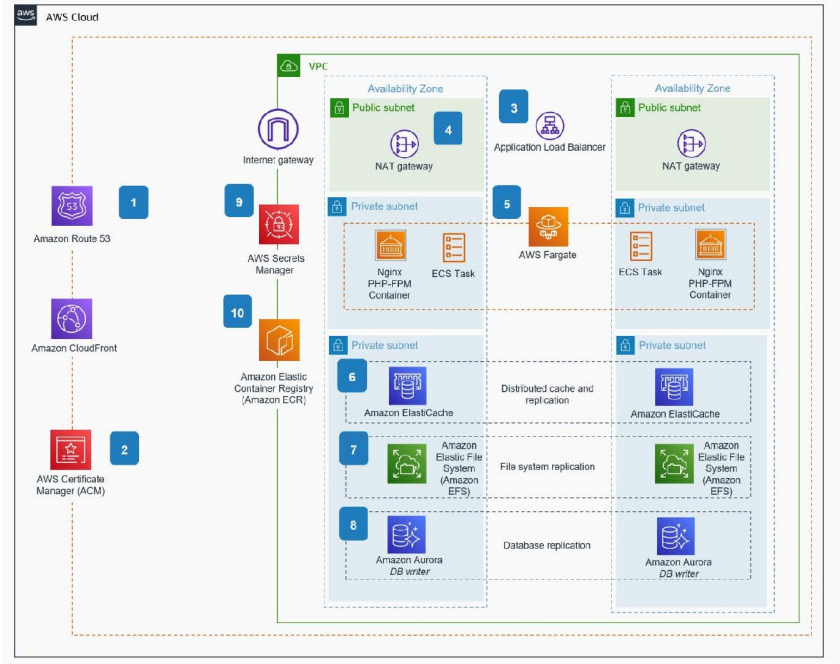


Fig. 4.1: Architectural Diagram for Moodle on AWS.

Example 4.15. **KISS** Arguably, this is the best-known principle; it applies to all ?aaS models and all requirements too. Despite its deceiving, nontechnical acronym, the Keep it Simple (Stupid) principle has seen wide applicability, in engineering as well as nontechnical fields. Examples of such applications include:

- Fashion designs. Figure 4.2 exemplifies.
- The RIP routing protocol in networking.
- The UDP protocol in networking.
- Lights at road intersections that do not take into account the volume of traffic at that intersection.

△

4.5.1.1 Types of principles

Note that principles can be tailored to the *specific ?aaS model*. One can also expect different principles to apply to different *business or technical requirements*. This latter point is also a source of confusion, as the following example demonstrates.

(a) KISS principle applied, cost is $55.

(b) Without KISS principle applied, cost is $138.

Fig. 4.2: KISS and fashion design.

Example 4.16. **Requirement or principle?** "Optimize for cost" can be considered a requirement objective, as discussed in Section 4.1.1.1. However, when one designs for a different requirement, say 24/7 availability, "Optimize for cost" can be considered a principle. One can use it select the cheapest among, say, two options to address availability. △

With this clarification in mind, we summarize from [4]:

Example 4.17. **10 Design Principles for AWS Cloud Architecture.** Note the use of verbs in most principles and the lack of verbs in some. Does this bother you?

1. Think Adaptive and Elastic
2. Treat servers as disposable resources
3. Automate Automate Automate
4. Implement loose coupling
5. Focus on services, not servers
6. Database is the base of it all
7. Be sure to remove single points of failure
8. Optimize for cost
9. Caching
10. AWS Cloud Architecture Security △

The AWS Certified Cloud Practitioner Wiki in [2] ups the ante by one and provides considerable details:

Example 4.18. **11 Design Principles for AWS Cloud Architecture. Core Principles.** Note the the lack of verbs in most. The Wiki explains.
1. Scalability
2. Disposable Resources Instead of Fixed Servers
3. Automation
4. Loose Coupling
5. Services, Not Servers
6. Databases
7. Managing Increasing Volumes of Data
8. Removing Single Points of Failure
9. Optimize for Cost
10. Caching
11. Security △

4.5.2 Five principles for cloud-native architecture

We quote from [5]:

> **5 principles for cloud-native architecture—what it is and how to master it**
>
> At Google Cloud, we often throw around the term 'cloud-native architecture' as the desired end goal for applications that you migrate or build on Google Cloud Platform (GCP). But what exactly do we mean by cloud-native? More to the point, how do you go about designing such a system?
>
> At a high level, *cloud-native architecture means adapting to the many new possibilities—but very different set of architectural constraints—offered by the cloud compared to traditional on-premises infrastructure*. Consider the high level elements that we as software architects are trained to consider:
>
> – The functional requirements of a system (what it should do, e.g 'process orders in this format...')
> – The non-functional requirements (how it should perform e.g. 'process at least 200 orders a minute')
> – Constraints (what is out-of-scope to change e.g. 'orders must be updated on our existing mainframe system').
>
> While the functional aspects don't change too much, the cloud offers, and sometimes requires, very different ways to meet non-functional requirements, and imposes very different architectural constraints. If architects fail to adapt their approach to these different constraints, the systems they architect are often fragile, expensive, and hard to maintain. A well-architected cloud native system, on the other hand, should be largely self-healing, cost efficient, and easily updated and maintained through Continuous Integration/Continuous Delivery (CI/CD).
>
> The good news is that cloud is made of the same fabric of servers, disks and networks that makes up traditional infrastructure. This means that almost all of the principles of good architectural design still apply for cloud-native architecture. However, some of the fundamental assumptions about how that fabric performs change when you're in the cloud. For instance, provisioning a replacement server can take weeks in traditional environments, whereas in the cloud, it takes seconds—your application architecture needs to take that into account.

In this post we set out five principles of cloud-native architecture that will help to ensure your designs take full advantage of the cloud while avoiding the pitfalls of shoe-horning old approaches into a new platform.

Principles for cloud-native architecture The principle of architecting for the cloud, a.k.a. cloud-native architecture, focuses on how to optimize system architectures for the unique capabilities of the cloud. Traditional architecture tends to optimize for a fixed, high-cost infrastructure, which requires considerable manual effort to modify. Traditional architecture therefore focuses on the resilience and performance of a relatively small fixed number of components. In the cloud however, such a fixed infrastructure makes much less sense because cloud is charged based on usage (so you save money when you can reduce your footprint) and it's also much easier to automate (so automatically scaling-up and down is much easier). Therefore, cloud-native architecture focuses on achieving resilience and scale though horizontal scaling, distributed processing, and automating the replacement of failed components. Let's take a look.

Principle 1: Design for automation Automation has always been a best practice for software systems, but cloud makes it easier than ever to automate the infrastructure as well as components that sit above it. Although the upfront investment is often higher, favouring an automated solution will almost always pay off in the medium term in terms of effort, but also in terms of the resilience and performance of your system. Automated processes can repair, scale, deploy your system far faster than people can. As we discuss later on, architecture in the cloud is not a one-shot deal, and automation is no exception—as you find new ways that your system needs to take action, so you will find new things to automate. Some common areas for automating cloud-native systems are:

- *Infrastructure:* Automate the creation of the infrastructure, together with updates to it, using tools like Google Cloud Deployment Manager or Terraform
- *Continuous Integration/Continuous Delivery*: Automate the build, testing, and deployment of the packages that make up the system by using tools like Google Cloud Build, Jenkins and Spinnaker. Not only should you automate the deployment, you should strive to automate processes like canary testing and rollback.
- *Scale up and scale down*: Unless your system load almost never changes, you should automate the scale up of the system in response to increases in load, and scale down in response to sustained drops in load. By scaling up, you ensure your service remains available, and by scaling down you reduce costs. This makes clear sense for high-scale applications, like public websites, but also for smaller applications with irregular load, for instance internal applications that are very busy at certain periods, but barely used at others. For applications that sometimes receive almost no traffic, and for which you can tolerate some initial latency, you should even consider scaling to zero (removing all running instances, and restarting the application when it's next needed).
- *Monitoring and automated recovery*: You should bake monitoring and logging into your cloud-native systems from inception. Logging and monitoring data streams can naturally be used for monitoring the health of the system, but can have many uses beyond this. For instance, they can give valuable insights into system usage and user behaviour (how many people are using the system, what parts they're using, what their average latency is, etc). Secondly, they can be used in aggregate to give a measure of overall system health (e.g., a disk is nearly full again, but is it filling faster than usual? What is the relationship between disk usage and service uptake? etc). Lastly, they are an ideal point for attaching automation. Now when that disk fills up, instead of just logging an error, you can also automatically resize the disk to allow the system to keep functioning.

Principle 2: Be smart with state Storing of 'state', be that user data (e.g., the items in the users shopping cart, or their employee number) or system state (e.g., how many instances of a job are running, what version of code is running in production), is the hardest aspect of architecting a distributed, cloud-native architecture. You should therefore architect your

system to be intentional about when, and how, you store state, and design components to be stateless wherever you can.

Stateless components are easy to:

- *Scale*: To scale up, just add more copies. To scale down, instruct instances to terminate once they have completed their current task.
- *Repair*: To 'repair' a failed instance of a component, simply terminate it as gracefully as possible and spin up a replacement.
- *Roll-back*: If you have a bad deployment, stateless components are much easier to roll back, since you can terminate them and launch instances of the old version instead.
- *Load-Balance across*: When components are stateless, load balancing is much simpler since any instance can handle any request. Load balancing across stateful components is much harder, since the state of the user's session typically resides on the instance, forcing that instance to handle all requests from a given user.

Principle 3: Favor managed services Cloud is more than just infrastructure. Most cloud providers offer a rich set of managed services, providing all sorts of functionality that relieve you of the headache of managing the backend software or infrastructure. However, many organizations are cautious about taking advantage of these services because they are concerned about being 'locked in' to a given provider. This is a valid concern, but managed services can often save the organization hugely in time and operational overhead.

Broadly speaking, the decision of whether or not to adopt managed services comes down to portability vs. operational overhead, in terms of both money, but also skills. Crudely, the managed services that you might consider today fall into three broad categories:

- *Managed open source or open source-compatible services*: Services that are managed open source (for instance Cloud SQL) or offer an open-source compatible interface (for instance Cloud Bigtable). This should be an easy choice since there are a lot of benefits in using the managed service, and little risk.
- *Managed services with high operational savings*: Some services are not immediately compatible with open source, or have no immediate open source alternative, but are so much easier to consume than the alternatives, they are worth the risk. For instance, BigQuery is often adopted by organizations because it is so easy to operate.
- *Everything else*: Then there are the hard cases, where there is no easy migration path off of the service, and it presents a less obvious operational benefit. You'll need to examine these on a case-by-case basis, considering things like the strategic significance of the service, the operational overhead of running it yourself, and the effort required to migrate away.

However, practical experience has shown that most cloud-native architectures favor managed services; the potential risk of having to migrate off of them rarely outweighs the huge savings in time, effort, and operational risk of having the cloud provider manage the service, at scale, on your behalf.

Principle 4: Practice defense in depth Traditional architectures place a lot of faith in perimeter security, crudely a hardened network perimeter with 'trusted things' inside and 'untrusted things' outside. Unfortunately, this approach has always been vulnerable to insider attacks, as well as external threats such as spear phishing. Moreover, the increasing pressure to provide flexible and mobile working has further undermined the network perimeter.

Cloud-native architectures have their origins in internet-facing services, and so have always needed to deal with external attacks. Therefore they adopt an approach of defense-in-depth by applying authentication between each component, and by minimizing the trust between those components (even if they are 'internal'). As a result, there is no 'inside' and 'outside'.

Cloud-native architectures should extend this idea beyond authentication to include things like rate limiting and script injection. Each component in a design should seek to protect itself from the other components. This not only makes the architecture very resilient, it also makes the resulting services easier to deploy in a cloud environment, where there may not be a trusted network between the service and its users.

Principle 5: Always be architecting One of the core characteristics of a cloud-native system is that it's always evolving, and that's equally true of the architecture. As a cloud-native architect, you should always seek to refine, simplify and improve the architecture of the system, as the needs of the organization change, the landscape of your IT systems change, and the capabilities of your cloud provider itself change. While this undoubtedly requires constant investment, the lessons of the past are clear: to evolve, grow, and respond, IT systems need to live and breathe and change. Dead, ossifying IT systems rapidly bring the organization to a standstill, unable to respond to new threats and opportunities.

The only constant is change In the animal kingdom, survival favors those individuals who adapt to their environment. This is not a linear journey from 'bad' to 'best' or from 'primitive' to 'evolved', rather everything is in constant flux. As the environment changes, pressure is applied to species to evolve and adapt. Similarly, cloud-native architectures do not replace traditional architectures, but they are better adapted to the very different environment of cloud. Cloud is increasingly the environment in which most of us find ourselves working, and failure to evolve and adapt, as many species can attest, is not a long term option.

The principles described above are not a magic formula for creating a cloud-native architecture, but hopefully provide strong guidelines on how to get the most out of the cloud. As an added benefit, moving and adapting architectures for cloud gives you the opportunity to improve and adapt them in other ways, and make them better able to adapt to the next environmental shift. Change can be hard, but as evolution has shown for billions of years, you don't have to be the best to survive—you just need to be able to adapt.

4.5.3 Principles in the Amazon Well-Architected Framework

See [1], pages 5 - 40.

This framework lists principles tailored to each pillar.

4.5.4 Principles in the Google Cloud Architecture Framework

—— This section is a stub for this semester. ——

See reference [6].

4.6 Best practices

4.6.1 What is a best practice?

A best practice is a canned solution for a BR/TR or set of BRs/TRs that worked. There is an abundance of best practices in industrial literature supplied by the cloud providers. They are created by architects or engineers in the provider organization and published to help (primarily inexperienced) comrades in consumer organizations.

- but no tradeoffs mentioned - be careful!

The purported benefits to those comrades are twofold:

- it saves time, since they do not have to create a solution from scratch;
- it saves "face" since "...Google told me to do it this way!".

There is a catch: since tradeoffs are seldom mentioned, applicability is questionable in a given situation.

4.6.2 Examples of best practices

Example 4.19. **Best practices for a cloud architect.** Reference [7] is full of best practice examples. They are organized on a per AWS Framework pillar. △

Example 4.20. **Best practices in SaaS.** We quote from [8]:

> **Architect for multi-tenant** A SaaS by default will service dozens to thousands of users, and more. Unless your intended customers will demand their own instance of your application, you should take advantage of architecting it to be a multi-tenant solution for both cost and efficiency.
>
> With a multi-tenancy SaaS solution, a single instance of the software serves all of the users. The underlying resources – database, server, and application – are shared. At the same time, user protected data is tagged and separated for security.
>
> A multi-tenant architecture is highly efficient since updates and maintenance are done to a single application, not handled individually. While the cost of cloud hosting increases as the size of needed resources, like storage and databases, increases, those costs grow much more slowly than the revenue from additional customers.

△

Example 4.21. **Best practices for a cloud admin for the provider.** We quote one such practice from the five presented in [9]:

> **Separate application and resource logging.** Application logging and resource logging should be distinct layers. Applications will often log their own conditions. Platform resource logs – for hosting and middleware – are always available. While the goal of centralization is to bring everything together, don't mix these two logging sources. Effective APM practices depend on isolating problems to applications or resources.

Admins should log information to determine the relationship between applications and their underlying resources. With the cloud and other virtual-hosting technologies, the connection between applications and hosting resources is soft because the application ”sees” the virtual resource. It is critical to map that application to the physical resource to make sense of the two log layers. This mapping is also critical to correlate issues, since applications that share physical resources won’t share the virtual resources.

△

4.7 Problem Section

Problems on business requirements

Problem 4.1. The customer is(n't) always right. Consider the generic business requirements of a cloud consumer we outlined in Section 4.1. Clearly, we need to make a tradeoff between cost and any other requirement. Do we need to make tradeoffs between

1. Availability 24/7 and performance?
2. Accommodating time-varying workloads and performance?
3. Availability 24/7 and avoiding "vendor lock-in"?
4. Availability 24/7, accommodating time-varying workloads and performance?

Explain clearly.

Problems on the process for creating an architecture

Problem 4.2. Is the process necessary? Consider the process we outlined in Section 4.4.1. It suggests that six steps are sufficient to create a "good" architecture.

Let's not question sufficiency. However, are the six steps really necessary?

1. Set $k = 1$
2. While $k < 7$
3. Remove step k from the process in Section 4.4.1.
4. Consider a new process without step k. Is it "good"? Justify your answer.
5. Set $k = k + 1$
6. Endwhile
7. Listen to `https://www.youtube.com/watch?v=MAe_w9a_IN8`

Problems on architecture principles

Problem 4.3. The KISS Principle. In Example 4.15, we have mentioned that the RIP routing protocol is an example of design that applied the KISS principle.

1. In what exact feature of RIP was the principle applied?
2. With respect to *this feature alone*, would you say that OSPF applied/did not apply the KISS principle?

Problem 4.4. The KISS Principle. In Example 4.15, we have mentioned that the RIP routing protocol is an example of design that applied the KISS principle.

1. In what exact feature of RIP was the principle applied?
2. With respect to *this feature alone*, would you say that BGP applied/did not apply the KISS principle?

Problem 4.5. The KISS Principle. Mention one aspect of the BGP protocol in which the KISS Principle was applied. Discuss what would be different in the protocol had the principle not been applied.

Problem 4.6. Internet Design Principles. Research principles that have been used in the design of the internet.

1. Describe one such principle.
2. Discuss how the principle was used in addressing an objective.

Problem 4.7. Principle 1. Design for automation. In Section 4.5.2, we presented 5 principles that a software architect can use.

1. Which ?aaS model(s) does the principle apply to? Justify your answer.
2. The principle mentions "Scale up and scale down" as an area for automating. Provide a specific idea on how to apply "Scale up and scale down" in Problem 2.9, page 36.
3. Discuss how the principle was used in addressing an objective.

Problem 4.8. Treat servers as disposable resources Principle. Consider the second principle in Example 4.16.

1. Describe the principle in your own words.
2. Identify the ?aaS model(s) you could apply this principle to.
3. Identify a business requirement you could apply this principle to.

Problems on the Amazon Well-Architected Framework

Problem 4.9. Check `https://aws.amazon.com/architecture/reference-architecture-diagra` it is a repository of reference architecture diagrams. You can filter by technology category or industry.

1. Select an architecture diagram of interest to you.
2. Describe, in your own words, the problem solved (what were the objectives?)
3. The diagram uses "standardized" AWS icons. Explain them.
4. List all the services involved.
5. Describe the services.
6. Comment on features of the design - e.g., is it scalable? highly available? secure? etc.

Problems on best practices

Problem 4.10. Best Practices for SaaS. Research Best Practices for SaaS.

1. What objective is the best practice addressing?
2. Does it give advice to an architect or implementor?

3. Does it discuss tradeoffs?

Problem 4.11. Best Practices for PaaS. Research Best Practices for PaaS.

1. What objective is the best practice addressing?
2. Does it give advice to an architect or implementor?
3. Does it discuss tradeoffs?

Problem 4.12. Best Practices for IaaS. Research Best Practices for IaaS.

1. What objective is the best practice addressing?
2. Does it give advice to an architect or implementor?
3. Does it discuss tradeoffs?

Problem 4.13. A best practice from the AWS Well-Architected Framework. Search [1] for a best practice. There are plenty! mention who benefits from the practice.

Problem 4.14. Best practices in Storage. Search [10] for a best practice. There are plenty! mention who benefits from the practice.

Huh?

Problem 4.15. Principles, principles. We argued in this chapter that the cloud computing industry is not very clear on what a principle is and how one can apply a principle while architecting.

1. Well, the networking industry is not better either. RFC1958, `https://datatracker.ietf.org/doc/html/rfc1958`, titled Architectural Principles of the Internet, is supposed to describe those principles. Can you tell how many principles are mentioned there? No need to dscribe them.
2. What is a parenting principle?
3. Describe two parenting principles.
4. Have you been subjected to a parenting principle in the past? If yes, comment on it. ☺

References

References marked as Handout H4.x are required reading material; you will find them in the required handouts binder in moodle. The rest are used in some form in the text or problems.

1. **Handout H4.1:** Amazon Well-Architected Framework https://docs.aws.amazon.com/wellarchitected/latest/framework/wellarchitected-framework.pdf
2. **Reference R4.1:** The Five Pillars of the Framework, https://aws-certified-cloud-practitioner.fandom.com/wiki/1.3_List_the_different_cloud_architecture_design_principles
3. **Reference R4.2:** AWS Reference Architecture for moodle, December 2021, https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/moodle-for-high-availability-on-AWS-ra.pdf
4. **Reference R4.3:** 10 Design Principles for AWS Cloud Architecture, https://www.botmetric.com/blog/aws-cloud-architecture-design-principles/
5. **Reference R4.4:** Tom Grey, "5 principles for cloud-native architecture—what it is and how to master it", June 2019 https://cloud.google.com/blog/products/application-development/5-principles-for-cloud-native-architecture-what-it-is-and-how-to-master-it
6. **Reference R4.5:** Google Cloud Architecture Framework https://cloud.google.com/architecture/framework
7. **Reference R4.6:** Cloud Architecture Best Practices: Using the Right Tools https://assets.ctfassets.net/6yom6slo28h2/45MPQ8h8jRJIWABx8KKEzM/ec275b96f203794288e36d04d32bf935/Kentik-Cloud-Architecture.pdf
8. **Reference R4.7:** Thomas Guidotti, "What Are the Best Practices for my SaaS Application?", https://www.curotec.com/insights/what-are-the-best-practices-for-my-saas-application/
9. **Reference R4.8:** 5 centralized logging best practices for cloud admins, https://searchcloudcomputing.techtarget.com/tip/5-centralized-logging-best-practices-for-cloud-admins
10. **Reference R4.9:** https://download.nutanix.com/solutionsDocs/RA-2006-Database-Workloads-on-Nutanix.pdf

Topic 5
Problems for the cloud automation engineer

Objectives: After reading this chapter, you should be able to: (a) describe the notions of automation, orchestration and integration, (b) identify how these notions differ, (c) list specific management tasks and describe corresponding tools, (d) describe the merits of Infrastructure as Code tools, (e) explain the fundamental concepts of and use the Kubernetes framework, (f) explain and use the Ansible framework, and, (g) explain and use the Openstack framework.

In this chapter, we discuss topics around the concepts of (a) cloud automation, (b) orchestration, and, (c) integration. They are related but different. Our focus will be only on tools that support these operations - kubernetes and Ansible in particular. Design issues around the first two topics were discussed in Chapter 6.

5.1 Cloud Automation, Orchestration and Integration (AOI)

—— This section is a stub for this semester. ——

In Section 1.3, we mentioned the role of Cloud administrator; this is the person in charge of executing the tasks involved in what we can vaguely call "cloud management". As is the case with all complex, large-scale systems, these are daunting tasks, to say the least. Manual execution is out of the question, for even the simplest-sounding ones. "Count the working physical servers in the datacenter" is an example of such a simple-sounding task.

5.1.1 Cloud Automation

Loosely speaking, Automation refers to executing a specific management task with minimal (ideally zero) involvement by the Cloud administrator hat. A management tool is used for this execution.

Example 5.1. **Automating creation of containers.** Docker △

Example 5.2. **Automating creation of VMs.** Openstack Nova. △

5.1.2 Cloud Orchestration

Loosely speaking, *orchestration is how you can automate a process or workflow that involves many steps across multiple disparate systems*; it goes beyond simple configurations.

We quote from [2]:

> **What is Cloud Orchestration?** Cloud Orchestration is the process of automating the tasks needed to manage connections and operations of workloads on private and public clouds. Cloud orchestration technologies integrate automated tasks and processes into a workflow to perform specific business functions.
>
> Cloud orchestration tools entail policy enforcement and ensure that processes have the proper permission to execute or connect to a workload. Typical cloud orchestration tasks are to provision or start server workloads, provision storage capacity as needed, and instantiate virtual machines (VMs) by orchestrating services, workloads, and resources in the cloud.
>
> Cloud automation and orchestration tools help to reduce the challenges organization have had deploying automation tools by eliminating islands of automation in favor of a cohesive, cloud-wide approach that encompasses both public cloud and private cloud components.
>
> **Why is Cloud Orchestration important?** The rapid adoption of containerized, microservices based applications that communicate via APIs has created the demand for automation of deploying and managing applications across the cloud. This increasing complexity has created the demand for cloud orchestration software that can manage the myriad dependencies across multiple clouds, with policy-driven security and management capabilities.
>
> As organizations increasingly adopt a hybrid cloud architecture, the need for both public cloud orchestration and hybrid cloud orchestration has continued to grow.
>
> Most importantly, cloud orchestration reduces the need for IT staff to manually handle automation tasks, freeing up resources for more productive works. This also reduces the opportunity for manual errors to occur. This lets organizations spend time on innovation, enabling accelerated deployment of applications across hybrid IT infrastructure, orchestrating various processes across domains and systems. The result is an improved experience for users and customers enterprise-wide.
>
> **What are Benefits of Cloud Orchestration?** Cloud orchestration simplifies automation across a hybrid cloud environment while ensuring that policies and security protocols are maintained in a dynamic, modern IT environment. Cloud orchestration reduces overall costs while accelerating delivery of services, automates management and coordination of complex hybrid environments, eliminates provisioning errors, and enables self-service provisioning of services without the need for IT intervention.

Cloud orchestration can also help prevent VM sprawl by enhancing the visibility into resource usage across clouds. Other high-level benefits include automating connections between workloads to ensure links are configured properly, and many cloud orchestration platforms do so by the use of a web-based portal that offers single pane of glass management organization-wide.In advanced organizations, developers and line-of-business workers can turn to cloud orchestration software as a self-service mechanism to deploy resources; administrators can use it to track the organization's reliance on various IT offerings and manage chargebacks.

Detailed benefits of cloud orchestration include:

• Improving efficiency of resource utilization, eliminate over-provisioning.
• Monitor, alert, and report on unexpected conditions to diagnose root cause.
• Simplify data integrations and automatically apply policies for governance and security.
• Manage dependencies across clouds to ensure proper execution of tasks.
• Integrate existing identity and access management systems as part of organization-wide security to ensure only authorized users and applications can access or modify automations.
• Eliminate need to build ad-hoc tools when new automations are required.
• Deliver workflow tools for managing and scheduling for IT and line of business users.
• Provide the bridge between clouds or between private and public environments.

What are Cloud Orchestration Models? Cloud Director orchestration models let you provision a ready environment to deploy virtual servers. Orchestration can be either single cloud or multi-cloud. In a single cloud model, multiple applications run on the same cloud service provider, which is a simpler setup. The more complicated, but also more powerful model is the multi-cloud setup. Here we have multiple applications, which are located on different cloud platforms, and multi-cloud orchestration interconnects them so they can perform as a single system, with the advantage of high redundancy

There are also three delivery models for cloud services in general, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS)

IaaS is most commonly utilized in cloud orchestration environments. IaaS providers offer network hardware, storage, and servers, as well as physical security in either a dedicated or multi-tenant environment. IaaS providers also offer virtualization services and orchestration tools that can streamline IT operations within their cloud or between multiple clouds.

PaaS providers also provide operating systems and middleware, and SaaS providers deliver applications only, through a web interface, typically on a subscription basis.

Cloud orchestration tools enable all these models to operate as one, providing automation across models and across clouds, but typically taking advantage of IaaS providers to automate the deployment process, reducing labor and resources required so they may focus on bottom-line generating functions.

What is Cloud Orchestration vs Cloud Automation? Orchestration is a superset of automation. Cloud orchestration goes beyond automation to also provide coordination between multiple automated tasks. Automation, on the other hand, has the goal of enabling one task, such as initiating a web server, to be repeated or iterated quickly with little manual intervention. Thus, cloud orchestration focuses on the entirely of the IT processes, automation on a single piece.

Practically, cloud orchestration and automation work hand in hand to ensure that cloud-based applications and services are deployed and delivered in a cohesive, functional, cloud environment that operates efficiently and cost effectively.

Cloud orchestration adds enterprise-wide functions such as redundancy, scalability, failover and fail-back, manages dependencies, and bundles them into a single package, greatly reducing the overall IT burden. Orchestration additionally offers enhanced visibility into the resources and processes in use, which can help prevent VM sprawl and help organizations track resource usage by department, business unit, or even by individual user. Orchestration enables automatic scaling of distributed applications, disaster recovery and business continuity, and inter-service configuration.

Cloud orchestration is increasingly important due to the growth of containerization, which facilitates scaling applications across clouds, both public and private. Tools such as VMware Tanzu help facilitate container orchestration across an organization's entire cloud presence.

5.1.3 Cloud Integration

5.1.4 The AOI problems we'll study

The main reference is [1].

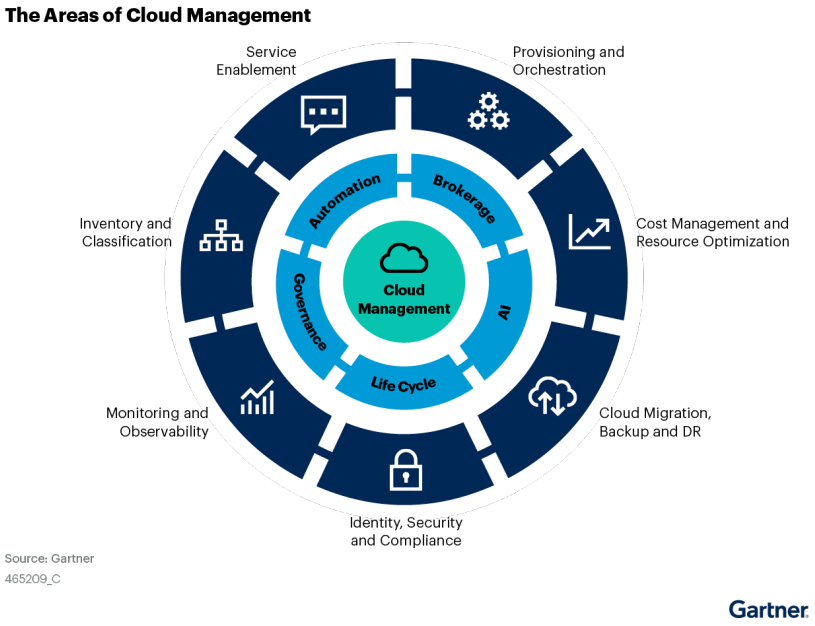


Fig. 5.1: The Areas of Cloud Management (from [1]).

5.2 Taxonomy of tools for AOI

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hard-

ware configuration. The IT infrastructure managed by this process comprises both physical equipment, such as bare-metal servers, as well as virtual machines, and associated configuration resources. The definitions may be in a version control system. In simpler terms, Infra as Code is literally what it says, i.e, converting Infrastructure to code (a program aka definition file).

- With Infra as code, you are no longer dealing with physical infrastructure directly instead you are working with the software interface, which handles the hardware. Think of a mobile phone, it is a hardware device but we use the software interface to interact (text, call, etc) with it.
- Instead of dealing with physical hardware configurations, you only have to write a code (machine-readable) that would fix the issues.

5.2.1 Tools for Infrastructure as Code

Infrastructure as a Code (IaC) can be a key attribute of enabling best practices in DevOps – Developers become more involved in defining configuration and Ops teams get involved earlier in the development process. Tools that utilize IaC bring visibility to the state and configuration of servers and ultimately provide the visibility to users within the enterprise, aiming to bring teams together to maximize their efforts.

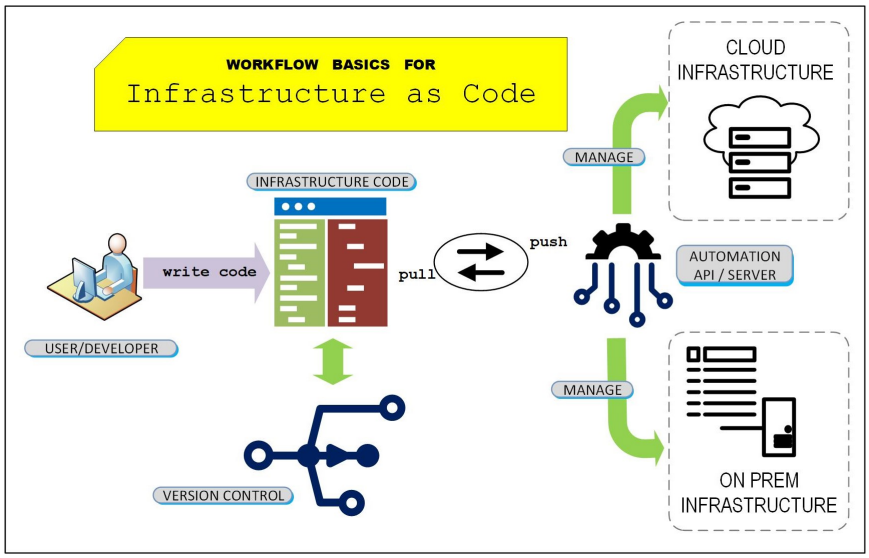


Fig. 5.2: Infrastructure as a Code usage

Tool	**Supplier**	Description
Terraform	Hashicorp	
Ansible	Redhat	Open Source suite of software tools that enables infrastructure as code
Cloud Formation	AWS	
Puppet	Puppet	Configuration Management tool
Chef	Progress	Configuration management tool
Google Cloud Deployment Manager	Google	
SaltStack	Open Source	event-driven IT automation, remote task execution, and configuration management
vCenter Configuration Manager (VCM)	VMware	Proprietary Configuration Management

Table 5.1: Tools for IaaS automation.

IaC is the future of large-scale computing. There is currently no alternative that provisions resources as effectively and consistently as IaC tools. It is crucial to any organization looking to scale up, but it can also be a time-saver for smaller organizations looking to start small.

Every infrastructure automation tool has its strengths, weaknesses, and learning curves. In choosing a tool, you'll want to find the one that best matches the needs of your project and the needs of your teams. Cloud-vendor built tools may provide the easiest way to get the ball rolling as you start your cloud migration, especially if a "one-stop shop" is what you're looking for.

That said, tools like Puppet, Chef, and Ansible have built tremendous reputations with DevOps practitioners managing infrastructures in all kinds of cloud environments. Their integrations prove it. In many cases architects select a combination of tools in order to deploy infrastructure in a more effective way.

5.2.2 Why use IaC tools

- **Speed**: Automation is faster than manually navigating an interface when you need to deploy and/or connect resources.
- **Reliability**: If your infrastructure is large, it becomes easy to misconfigure a resource or provision services in the wrong order. With IaC, the resources are always provisioned and configured exactly as declared.
- **Scaling** Once an IaC is defined, scaling into many systems has reduced cost and time investment.

- **A global solution** The nature of IaC means that any system can be used with it, whether it is in the Cloud or local, VM or not.
- **Better testing** A development team that employs IaC can always test on systems identical to production. Consistent systems increase the efficacy of testing and reduce surprises at later stages.
- **Version Control** Like code, IaC can be version controlled, so when something environmental causes the system to respond incorrectly, it can be rolled back and investigated.
- **Prevent configuration drift**: Configuration drift occurs when the configuration that provisioned your environment no longer matches the actual environment. (See 'Immutable infrastructure' below.)
- **Support experimentation, testing, and optimization**: Because Infrastructure as Code makes provisioning new infrastructure so much faster and easier, you can make and test experimental changes without investing lots of time and resources; and if you like the results, you can quickly scale up the new infrastructure for production.
- **Documentation** IaC is a self-documenting practice, as the definitions of the IaC serve to inform everything about the system's configuration.
- **Cost** All of the above lead to a reduction in price in the long run. As your needs scale, the returns on investment grow. The earlier you deploy using IaC, the better the rewards.

While there are many benefits to IaC, it is essential to keep in mind that there is a steep learning curve. Changing into IaC can be challenging if you already have systems running, which is another reason why early is better. The advantages far outweigh the challenges, much like setting up a CI/CD pipeline.

5.2.3 Approaches to Implement IaC

- Declarative approach — In this, the focus is on the end goal i.e the final desired infrastructure state. The system will take care of the process for you.
- Imperative approach — In this, the focus is on the process or steps required to accomplish the end goal.

Many IaC tools use a declarative approach and will automatically provision the desired infrastructure. If you make changes to the desired state, a declarative IaC tool will apply those changes for you. An imperative tool will require you to figure out how those changes should be applied.

IaC tools are often able to operate in both approaches, but tend to prefer one approach over the other.

5.2.4 Cloud Agnostic Tools

5.2.4.1 The Terraform tool

We quote from [3]:

What is Terraform? This guide highlights everything you need to know about Terraform—a tool that allows programmers to build, change, and version infrastructure safely and efficiently.

Terraform is an open source "Infrastructure as Code" tool, created by HashiCorp.

A declarative coding tool, Terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired "end-state" cloud or on-premises infrastructure for running an application. It then generates a plan for reaching that end-state and executes the plan to provision the infrastructure.

Because Terraform uses a simple syntax, can provision infrastructure across multiple cloud and on-premises data centers, and can safely and efficiently re-provision infrastructure in response to configuration changes, it is currently one of the most popular infrastructure automation tools available. If your organization plans to deploy a hybrid cloud or multicloud environment, you'll likely want or need to get to know Terraform.

Why Terraform? There are a few key reasons developers choose to use Terraform over other Infrastructure as Code tools:

• **Open source**: Terraform is backed by large communities of contributors who build plugins to the platform. Regardless of which cloud provider you use, it's easy to find plugins, extensions, and professional support. This also means Terraform evolves quickly, with new benefits and improvements added consistently.

• **Platform agnostic**: Meaning you can use it with any cloud services provider. Most other IaC tools are designed to work with single cloud provider.

• **Immutable infrastructure**: Most Infrastructure as Code tools create mutable infrastructure, meaning the infrastructure can change to accommodate changes such as a middleware upgrade or new storage server. The danger with mutable infrastructure is configuration drift—as the changes pile up, the actual provisioning of different servers or other infrastructure elements 'drifts' further from the original configuration, making bugs or performance issues difficult to diagnose and correct. Terraform provisions immutable infrastructure, which means that with each change to the environment, the current configuration is replaced with a new one that accounts for the change, and the infrastructure is reprovisioned. Even better, previous configurations can be retained as versions to enable rollbacks if necessary or desired.

Terraform modules Terraform modules are small, reusable Terraform configurations for multiple infrastructure resources that are used together. Terraform modules are useful because they allow complex resources to be automated with re-usable, configurable constructs. Writing even a very simple Terraform file results in a module. A module can call other modules—called child modules—which can make assembling configuration faster and more concise. Modules can also be called multiple times, either within the same configuration or in separate configurations.

Terraform providers Terraform providers are plugins that implement resource types. Providers contain all the code needed to authenticate and connect to a service—typically from a public cloud provider—on behalf of the user. You can find providers for the cloud platforms and services you use, add them to your configuration, and then use their resources to provision infrastructure. Providers are available for nearly every major cloud provider, SaaS offering, and more, developed and/or supported by the Terraform community or individual organizations. Refer to the Terraform documentation (link resides outside ibm.com) for a detailed list.

Terraform vs. Kubernetes Sometimes, there confusion between Terraform and Kubernetes and what they actually do. The truth is that they are not alternatives and actually work effectively together.

Kubernetes is an open source container orchestration system that lets developers schedule deployments onto nodes in a compute cluster and actively manages containerized workloads to ensure that their state matches the users' intentions. Terraform, on the other hand, is an Infrastructure as Code tool with a much broader reach, letting developers automate complete infrastructure that spans multiple public clouds and private clouds.

Terraform can automate and manage Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), or even Software-as-a-Service (SaaS) level capabilities and build all these resources across all those providers in parallel. You can use Terraform to automate the provisioning of Kubernetes—particularly managed Kubernetes clusters on cloud platforms—and to automate the deployment of applications into a cluster.

Terraform vs. Ansible Terraform and Ansible are both Infrastructure as Code tools, but there are a couple significant differences between the two:

• While Terraform is purely a declarative tool (see above), Ansible combines both declarative and procedural configuration. In procedural configuration, you specify the steps, or the precise manner, in which you want to provision infrastructure to the desired state. Procedural configuration is more work but it provides more control.

• Terraform is open source; Ansible is developed and sold by Red Hat.

IBM and Terraform IBM Cloud Schematics is IBM's free cloud automation tool based on Terraform. IBM Cloud Schematics allows you to fully manage your Terraform-based infrastructure automation so you can spend more time building applications and less time building environments.

5.2.5 Cloud-specific Tools

5.2.5.1 AWS CloudFormation

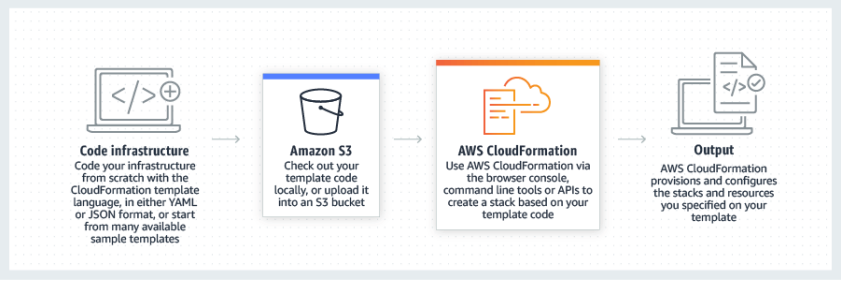


Fig. 5.3: AWS CloudFormation.

AWS CloudFormation allows you to manage infrastructure and automate any deployments using code. The main difference comes down to how intimate Cloud-

Formation is to AWS in that it only works with AWS IaC. However, it makes up for this by being integrated with the entire platform.

You can write CloudFormation templates in both YAML and JSON, which you can use to make managing, scaling, and automating AWS resources fast and straightforward. Furthermore, you can preview all the changes before deployment, which helps you visualize the impact a set of changes will have on your resources, services, and dependencies.

CloudFormation also offers Rollback Triggers that allow you to restore infrastructure to a previous state, guaranteeing controlled deployments in case of any mistakes or issues.

This tool's close relationship with AWS enables infrastructure stacks to be deployed in several regions and accounts using the same CloudFormation template.

5.2.5.2 Azure Resource Manager

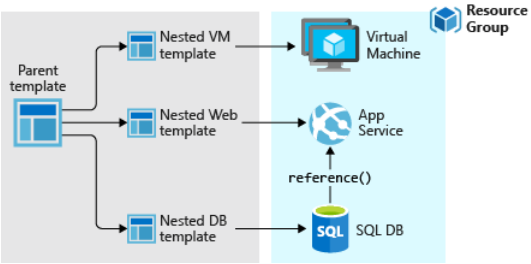


Fig. 5.4: Azure Resource Manager.

Azure Resource Manager is Microsoft's tool to manage Infrastructure in its platform. It uses the Azure Resource Manager template (ARM templates) to handle dependencies and infrastructure. For example, you can organize your resources into groups, delete them, control access levels to resources, just to name a few.

Controlling access to services and resources is made easy when using Azure, as it supports Role-Based Access Control (RBAC) natively. On the other hand, you can finetune the scope of access with management groups, subscriptions, and resource groups. Additionally, lower levels of hierarchy inherit settings from higher levels, ensuring that policy enforced by higher levels is applied at all desired lower level groups and resources.

ARM offers templates that can deploy resources in parallel, making it possible for faster deployments. Furthermore, the system comes with great organization tools, letting you attribute tags to resources, organize your groups, and check the costs of any resource sharing a specific tag.

5.2.5.3 Google Cloud Deployment Manager

Cloud Deployment Manager is Google's infrastructure deployment service. It uses declarative language to automate the management, creation, provisioning, and configuration of Google Cloud Platform resources. With it, you can use YAML or Python scripts to manage resources alone.

On conveniently organized resource groups, you can use this code in the future to produce equally consistent deployments. It also enables you to preview the impact of all your changes before they're applied. If the need arises, you can use the built-in console to check your current deployments as well.

However, what sets the Deployment Manager apart from the other Infrastructure as Code tools in this list is how deeply integrated it is to Google's ecosystem. Essentially, it offers UI support inside the developer's console, making it faster to visualize the architecture of deployments. In addition, being native to the platform, Deployment Manager requires no additional configuration software, and no additional cost is charged for it.

5.3 Deployments

Historically, developers brought applications offline when deploying changes and updates, resulting in downtime. Now, continuous integration and continuous deployment (CI/CD) pipelines that automate application build, test, and deployment. CI/CD piplines keep environments up as much as possible, and speed up the deployment process. Deploying an application or updating an environment can still cause downtime and other issues.

For most enterprises, especially those with an up-to-date application architecture and an established CI/CD pipeline, the goal is to deploy applications and features at any time with no noticeable effect on the end user. This requires deploying changes into production environments, rapidly and safely, without:

- Interrupting users who may be performing critical tasks
- Relying on your systems for mission-critical processes

Your application and deployment architecture plays a key role in reducing deployment downtime. Generally, your environment should meet the following requirements for both the canary and blue-green deployment methods:

- A deployment pipeline that can build, test, and deploy to specific environments
- Multiple application nodes or containers distributed behind a load balancer

An application that is stateless, allowing any nodes in the cluster to serve requests at any time

When you make changes to your application, they should be non-destructive to your data layer. This means you should make columns in your datasets optional or

nullable. Don't rename or reuse columns for different purposes. Adopting this non-destructive approach in your data layer enables you to roll back changes or unwind features if something goes wrong.

5.3.1 Basic Deployment

In a basic deployment, all nodes within a target environment are updated at the same time with a new service or artifact version. Because of this, basic deployments are not outage-proof and they slow down rollback processes or strategies. Of all the deployment strategies shared, it is the riskiest.

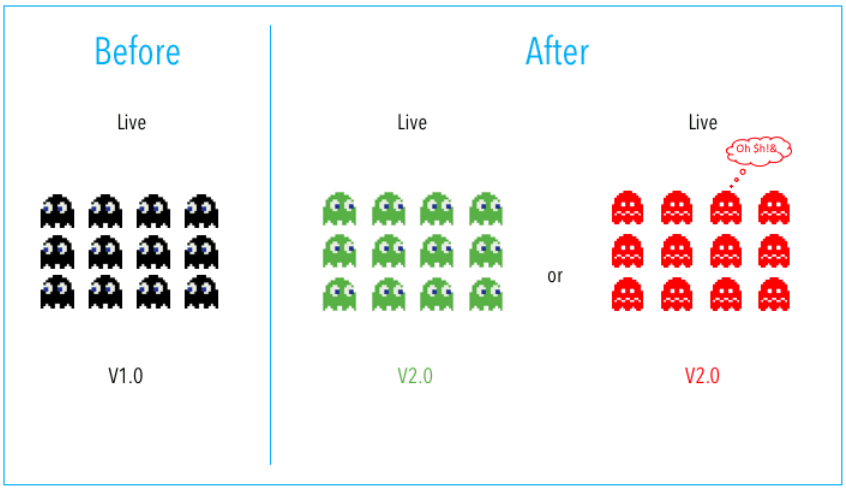


Fig. 5.5: Basic Deployment

5.3.2 Multiservice Deployment

In a multi-service deployment, all nodes within a target environment are updated with multiple new services simultaneously. This strategy is used for application services that have service or version dependencies, or if you're deploying off-hours to resources that are not in use.

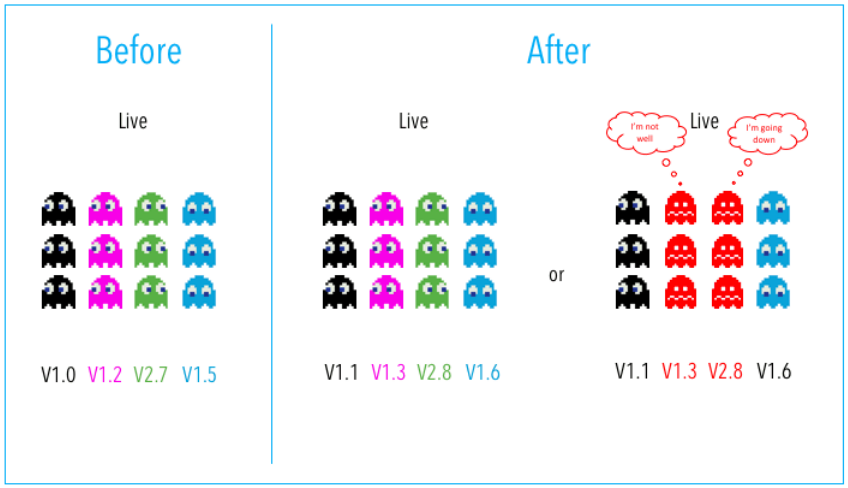


Fig. 5.6: Multi Service Deployment.

5.3.3 Blue/Green Deployment

Blue-green deployment splits your application environment into two equally-resourced sections, Blue and Green. You serve the current application on one half of your environment (Blue) using your load balancer to direct traffic. You can then deploy your new application to the other half of your environment (Green) without affecting the blue environment.

Using your load balancers to direct traffic keeps your blue environment running seamlessly for production users while you test and deploy to your green environment. When your deployment and testing are successful, you can switch your load balancer to target your green environment with no perceptible change for your users.

5.3.4 Canary Deployment

Canary deployment works similarly to blue-green deployment, but uses a slightly different method. Instead of another full environment waiting to be switched over once deployment is finished, canary deployments cut over just a small subset of servers or nodes first, before finishing the others.

In software engineering, canary deployment is the practice of making staged releases. We roll out a software update to a small part of the users first, so they may test it and provide feedback. Once the change is accepted, the update is rolled out to the rest of the users.

Fig. 5.7: Basic Deployment

There are many ways to configure your environment for canary deployments. The simplest way is to set up your environment behind your load balancer as normal, but keep an additional node or server or two (depending on the size of your application) as an unused spare. This spare node or server group is the deployment target for your CI/CD pipeline. Once you build, deploy, and test this node, you add it back into your load balancer for a limited time for a limited group of people. This allows you to make sure changes are successful before you repeat the process with the other nodes in your cluster.

The other option for configuring a canary deployment is to use a development pattern called feature toggles. Feature toggles (sometimes called feature flags) work by building and deploying your changes to an application controlled by a configuration that switches those changes on. You take a node out of your cluster, deploy, and add it back in — but without needing to test or control anything through the load balancer. Then, when all the nodes are updated, you toggle the feature on for a number of users before rolling the feature out to everyone.

The downside to this method, however, is the development time and cost of modifying your application to support feature toggles. Depending on the age and size of your application, developing this feature could be fairly complex or nearly impossible

5.3.5 What Deployment Strategy is the best?

Most teams use blue-green or canary deployments for mission-critical web applications.

Customers have minimal to little business impact when migrating from the blue-green deployment strategy to a canary deployment strategy.

It's also common for teams to create their strategy based on combining the strategies. For example, some customers will do multi-service canary deployments.

5.3.6 A/B Testing

In A/B testing, different versions of the same service run simultaneously as "experiments" in the same environment for a period of time. Experiments are either controlled by feature flags toggling, A/B testing tools, or through distinct service deployments. It is the experiment owner's responsibility to define how user traffic is routed to each experiment and version of an application. Commonly, user traffic is routed based on specific rules or user demographics to perform measurements and comparisons between service versions. Target environments can then be updated with the optimal service version.

The biggest difference between A/B testing and other deployment strategies is that A/B testing is primarily focused on experimentation and exploration. While other deployment strategies deploy many versions of a service to an environment with the immediate goal of updating all nodes with a specific version, A/B testing is about testing multiple ideas vs. deploying one specific tested idea.

5.4 Kubernetes

Το Αλφα και το Ωμεγα οσον αφορα τον Κυβερνητη θα το βρουμε στο Kubernetes Documentation site. Περιεχει τα παντα, αλλα για καποιον που ειναι 'αρχαριος' στον χωρο, It's all Greek to me.

Everything we need to know in order to (deeply) understand Kubernetes (aka k8s) as well as experiment with it can be found in the Kubernetes Documentation site, `https://kubernetes.io/docs/home/`. The site contains enormous amounts of material, from Getting started, to concepts, to installing, getting trained, using, up to even contributing to k8s...

For starters, stopping short of relying on free "k8s for dummies" references or costly books, we'll use the VMWare summary about k8s in reference [5]. This will contain material we can use in both the lectures as well as project/lab exercises.

5.4.1 Overview of Kubernetes

We quote from the Kubernetes Documentation site `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`:

> **What is Kubernetes?** Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
>
> The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s". Google open-sourced the k8s project in 2014. k8s combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.
>
> **Going back in time** Let's take a look at why k8s is so useful by going back in time.

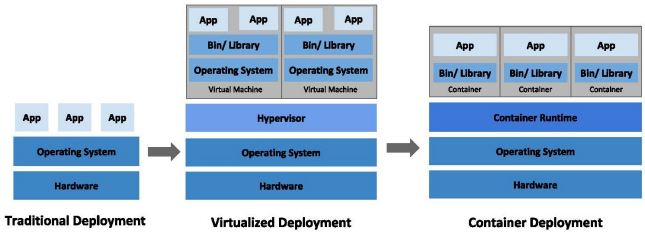


Fig. 5.8: Evolution of application deployment.

Traditional deployment era: Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

Virtualized deployment era: As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment era: Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Containers have become popular because they provide extra benefits, such as:

- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability: not only surfaces OS-level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- Resource isolation: predictable application performance.
- Resource utilization: high efficiency and density.

Why you need k8s and what it can do Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how k8s comes to the rescue! k8s provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, k8s can easily manage a canary deployment for your system.

k8s provides you with:

- *Service discovery and load balancing.* k8s can expose a container using the DNS name or using their own IP address. If traffic to a container is high, k8s is able to load balance and distribute the network traffic so that the deployment is stable.
- *Storage orchestration.* k8s allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- *Automated rollouts.* and rollbacks You can describe the desired state for your deployed containers using k8s, and it can change the actual state to the desired state at a controlled rate. For example, you can automate k8s to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- *Automatic bin packing.* You provide k8s with a cluster of nodes that it can use to run containerized tasks. You tell k8s how much CPU and memory (RAM) each container needs. k8s can fit containers onto your nodes to make the best use of your resources.
- *Self-healing.* k8s restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

- *Secret and configuration management.* k8s lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

What k8s is not k8s is not a traditional, all-inclusive PaaS (Platform as a Service) system. Since k8s operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions. However, k8s is not monolithic, and these default solutions are optional and pluggable. k8s provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

k8s:

- Does not limit the types of applications supported. k8s aims to support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads. If an application can run in a container, it should run great on k8s.
- Does not deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures and preferences as well as technical requirements.
- Does not provide application-level services, such as middleware (for example, message buses), data-processing frameworks (for example, Spark), databases (for example, MySQL), caches, nor cluster storage systems (for example, Ceph) as built-in services. Such components can run on k8s, and/or can be accessed by applications running on k8s through portable mechanisms, such as the Open Service Broker.
- Does not dictate logging, monitoring, or alerting solutions. It provides some integrations as proof of concept, and mechanisms to collect and export metrics.
- Does not provide nor mandate a configuration language/system (for example, Jsonnet). It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.
- Additionally, k8s is not a mere orchestration system. In fact, it eliminates the need for orchestration. The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C. In contrast, k8s comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state. It shouldn't matter how you get from A to C. Centralized control is also not required. This results in a system that is easier to use and more powerful, robust, resilient, and extensible.

5.4.2 Basic notions and terminology

In order to provide a sufficient description of the components and architecture of a k8s system, we need to first define certain terms.

- **Pod.** Pods are the smallest deployable units of computing that you can create and manage in k8s. See `https://kubernetes.io/docs/concepts/workloads/pods/` for more details.

  A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the

containers. A Pod's contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host.
Pods in a k8s cluster are used in two main ways:
Pods that run a single container. The "one-container-per-Pod" model is the most common k8s use case; in this case, you can think of a Pod as a wrapper around a single container; k8s manages Pods rather than managing the containers directly.
Pods that run multiple containers that need to work together. A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit of service—for example, one container serving data stored in a shared volume to the public, while a separate sidecar container refreshes or updates those files. The Pod wraps these containers, storage resources, and an ephemeral network identity together as a single unit.

- **ReplicaSet.** A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods. See `https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/` for more details.
- **Deployment.** A Deployment provides declarative updates for Pods and ReplicaSets. See `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/` for more details.
  You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.
  The following are typical use cases for Deployments:
  Create a Deployment to rollout a ReplicaSet. The ReplicaSet creates Pods in the background. Check the status of the rollout to see if it succeeds or not.
  Declare the new state of the Pods by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new ReplicaSet updates the revision of the Deployment.
  Rollback to an earlier Deployment revision if the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.
  Scale up the Deployment to facilitate more load.
  Pause the rollout of a Deployment to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.
  Troubleshoot rollouts. Use the status of the Deployment as an indicator that a rollout has stuck.
  Clean up older ReplicaSets that you don't need anymore.
- **Controller.** In k8s, controllers are control loops that watch the state of your cluster, then make or request changes where needed. Each controller tries to move the

current cluster state closer to the desired state. See https://kubernetes.io/docs/concepts/architecture/controller/ for more details.
A controller tracks at least one k8s resource type. These objects have a spec field that represents the desired state. The controller(s) for that resource are responsible for making the current state come closer to that desired state.
The controller might carry the action out itself; more commonly, in k8s, a controller will send messages to the API server that have useful side effects.

- **Service.** A k8s service is a logical abstraction for a deployed group of pods in a cluster (which all perform the same function). See https://www.vmware.com/topics/glossary/content/kubernetes-services.html for more details.
  Since pods are ephemeral, a service enables a group of pods, which provide specific functions (web services, image processing, etc.) to be assigned a name and unique IP address (clusterIP). As long as the service is running that IP address, it will not change. Services also define policies for their access.
  k8s services connect a set of pods to an abstracted service name and IP address. Services provide discovery and routing between pods. For example, services connect an application front-end to its backend, each of which running in separate deployments in a cluster. Services use labels and selectors to match pods with other applications. The core attributes of a k8s service are:
  A label selector that locates pods
  The clusterIP IP address and assigned port number
  Port definitions
  Optional mapping of incoming ports to a targetPort
- **Objects.** k8s objects are persistent entities in the k8s system. k8s uses these entities to represent the state of your cluster. See https://kubernetes.io/docs/concepts/overview/working-with-objects/ for more details.
  Specifically, they can describe:
  What containerized applications are running (and on which nodes)
  The resources available to those applications
  The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance
  A k8s object is a "record of intent"–once you create the object, the k8s system will constantly work to ensure that object exists. By creating an object, you're effectively telling the k8s system what you want your cluster's workload to look like; this is your cluster's desired state.
  To work with k8s objects–whether to create, modify, or delete them–you'll need to use the k8s API. When you use the kubectl command-line interface, for example, the CLI makes the necessary k8s API calls for you. You can also use the k8s API directly in your own programs using one of the Client Libraries.
  Object spec and status. Almost every k8s object includes two nested object fields that govern the object's configuration: the object spec and the object status. For objects that have a spec, you have to set this when you create the object, providing a description of the characteristics you want the resource to have: its desired state.

The status describes the current state of the object, supplied and updated by the k8s system and its components. The k8s control plane continually and actively manages every object's actual state to match the desired state you supplied.
For example: in k8s, a Deployment is an object that can represent an application running on your cluster. When you create the Deployment, you might set the Deployment spec to specify that you want three replicas of the application to be running. The k8s system reads the Deployment spec and starts three instances of your desired application–updating the status to match your spec. If any of those instances should fail (a status change), the k8s system responds to the difference between spec and status by making a correction–in this case, starting a replacement instance.

5.4.3 An (incomplete list of) BRs and TRs Kubernetes addresses

k8s addresses BRs/TRs for three hats: application developers, administrators and architects.

5.4.3.1 A few BRs

Arguably, the main BR that sparked the k8s project at Google can be stated simply as:

Simplify application deployment

We can also cite related BRs:

1. Reduce application deployment times
2. Reduce operation costs
3. Simplify management tasks

5.4.3.2 A few TRs

The BRs of the previous section can be expanded into a plethora of more specific TRs. We mention a few below:

1. Monitor the health of an application; replace one that fails
2. Balance traffic among containers that run the same application
3. Avoid allocation of resources for peak usage
4. Adapt to varying workload needs
5. Automate discovery of an application via DNS

5.4.4 The main components

We quote from `https://kubernetes.io/docs/concepts/overview/components/`:

k8s Components When you deploy k8s, you get a cluster.

A k8s cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

This document outlines the various components you need to have for a complete and working k8s cluster. Figure 5.9 depicts these components.

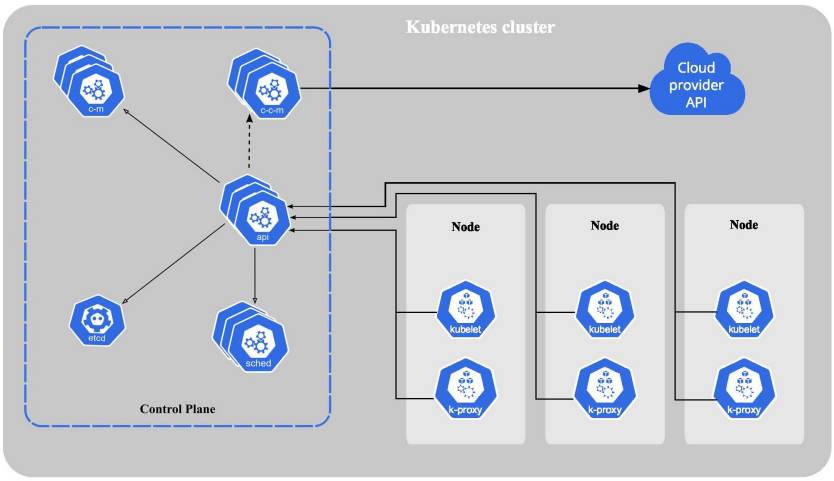


Fig. 5.9: The components of a k8s cluster.

5.4.4.1 Control Plane Components

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Creating Highly Available clusters with kubeadm for an example control plane setup that runs across multiple machines.

kube-apiserver The API server is a component of the k8s control plane that exposes the k8s API. The API server is the front end for the k8s control plane.

The main implementation of a k8s API server is kube-apiserver. kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

etcd Consistent and highly-available key value store used as k8s' backing store for all cluster data.

If your k8s cluster uses etcd as its backing store, make sure you have a back up plan for the data.

You can find in-depth information about etcd in the official documentation.

kube-scheduler Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

kube-controller-manager Control plane component that runs controller processes.

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

Node controller: Responsible for noticing and responding when nodes go down.

Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

EndpointSlice controller: Populates EndpointSlice objects (to provide a link between Services and Pods).

ServiceAccount controller: Create default ServiceAccounts for new namespaces.

cloud-controller-manager A k8s control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster. The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running k8s on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

The following controllers can have cloud provider dependencies:

Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding

Route controller: For setting up routes in the underlying cloud infrastructure

Service controller: For creating, updating and deleting cloud provider load balancers

5.4.4.2 Node Components

Node components run on every node, maintaining running pods and providing the k8s runtime environment.

kubelet An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by k8s.

kube-proxy kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the k8s Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

Container runtime The container runtime is the software that is responsible for running containers.

k8s supports container runtimes such as containerd, CRI-O, and any other implementation of the k8s CRI (Container Runtime Interface).

5.4.4.3 Addons

Addons use k8s resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

Selected addons are described below; for an extended list of available addons, please see Addons.

DNS While the other addons are not strictly required, all k8s clusters should have cluster DNS, as many examples rely on it.

Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for k8s services.

Containers started by k8s automatically include this DNS server in their DNS searches.

Web UI (Dashboard) Dashboard is a general purpose, web-based UI for k8s clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

Container Resource Monitoring Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.

Cluster-level Logging A cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.

5.4.5 Networking in k8s

We quote from Reference [12], Chapter 10, Advanced Kubernetes Networking;

5.4.5.1 Understanding the k8s networking model

The k8s networking model is based on a flat address space. All pods in a cluster can directly see each other. Each pod has its own IP address. There is no need to configure any NAT. In addition, containers in the same pod share their pod's IP address and can communicate with each other through localhost. This model is pretty opinionated, but, once set up, it simplifies life considerably both for developers and administrators. It makes it particularly easy to migrate traditional network applications to k8s. A pod represents a traditional node and each container represents a traditional process.

5.4.5.2 Intra-pod communication (container to container)

A running pod is always scheduled on one (physical or virtual) node. That means that all the containers run on the same node and can talk to each other in various ways, such as the local filesystem, any IPC mechanism, or using localhost and well-known ports. There is no danger of port collision between different pods because each pod has its own IP address, and when a container in the pod uses localhost, it applies to the pod's IP address only. So, if container 1 in pod 1 connects to port 1234, which container 2 listens to on pod 1, it will not conflict with another container in pod 2 running on the same node that also listens on port 1234. The only caveat is that if you're exposing ports to the host then you should be careful about pod-to-node affinity. This can be handled using several mechanisms, such as DaemonSet and pod anti-affinity.

5.4.5.3 Inter-pod communication (pod to pod)

Pods in k8s are allocated a network-visible IP address (not private to the node). Pods can communicate directly without the aid of network address translation, tunnels, proxies, or any other obfuscating layer. Well-known port numbers can be used for a configuration-free communication scheme. The pod's internal IP address is the same as its external IP address that other pods see (within the cluster network; not exposed to the outside world). This means that standard naming and discovery mechanisms such as DNS work out of the box.

5.4.5.4 Pod-to-service communication

Pods can talk to each other directly using their IP addresses and well-known ports, but that requires the pods to know each other's IP addresses. In a k8s cluster, pods can be destroyed and created constantly. The service provides a layer of indirection that is very useful because the service is stable even if the set of actual pods that respond to requests is ever-changing. In addition, you get automatic, highly-available load balancing because the Kube-proxy on each node takes care of redirecting traffic to the correct pod:

5.4.5.5 External access

Eventually, some containers need to be accessible from the outside world. The pod IP addresses are not visible externally. The service is the right vehicle, but external access typically requires two redirects. For example, cloud provider load balancers are Kubernetes-aware, so they can't direct traffic to a particular service directly to a node that runs a pod that can process the request. Instead, the public load balancer just directs traffic to any node in the cluster and the Kube-proxy on that node will redirect again to an appropriate pod if the current node doesn't run the necessary pod.

The following diagram shows how all that the external load balancer on the right side does is send traffic to all nodes that reach the proxy, which takes care of further routing, if it's needed:

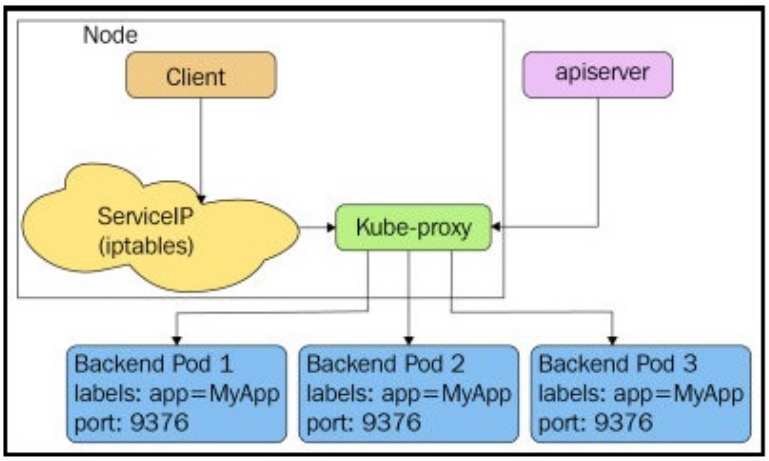


Fig. 5.10: Pod-to-service communication.

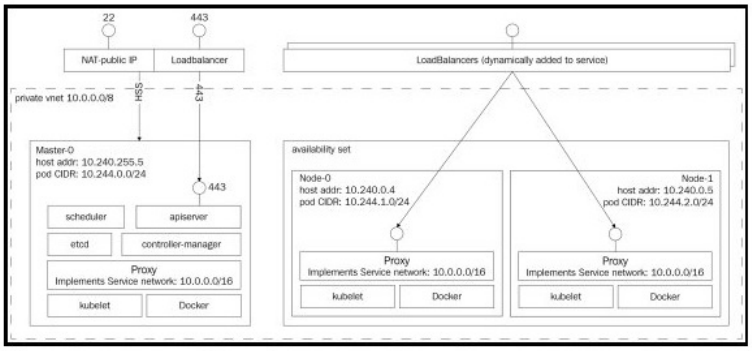


Fig. 5.11: External access.

5.4.5.6 Lookup and discovery

In order for pods and containers to communicate with each other, they need to find each other. There are several ways for containers to locate other containers or announce themselves. There are also some architectural patterns that allow containers to interact indirectly. Each approach has its own pros and cons.

5.4.5.7 Self-registration

We've mentioned self-registration several times. Let's understand exactly what it means. When a container runs, it knows its pod's IP address. Each container that wants to be accessible to other containers in the cluster can connect to some registration service and register its IP address and port. Other containers can query the registration service for the IP addresses and port of all registered containers and connect to them. When a container is destroyed (gracefully), it will unregister itself. If a container dies ungracefully then some mechanism needs to be established to detect that. For example, the registration service can periodically ping all registered containers, or the containers are required periodically to send a keepalive message to the registration service. The benefit of self-registration is that once the generic registration service is in place (no need to customize it for different purposes), there is no need to worry about keeping track of containers. Another huge benefit is that containers can employ sophisticated policies and decide to unregister temporarily if they are unavailable because of local conditions, such as if a container is busy and doesn't want to receive any more requests at the moment. This sort of smart and decentralized dynamic load balancing can be very difficult to achieve globally. The downside is that the registration service is yet another non-standard component that containers need to know about in order to locate other containers.

5.4.5.8 Services and endpoints

k8s services can be considered as a registration service. Pods that belong to a service are registered automatically based on their labels. Other pods can look up the endpoints to find all the service pods or take advantage of the service itself and directly send a message to the service that will get routed to one of the backend pods. Although most of the time, pods will just send their message to the service itself, which will forward it to one of the backing pods.

5.4.5.9 Loosely coupled connectivity with queues

—— This section is a stub for this semester. ——

What if containers can talk to each other without knowing their IP addresses and ports or even service IP addresses or network names? What if most of the communication can be asynchronous and decoupled? In many cases, systems can be composed of loosely coupled components that are not only unaware of the identities of other components, but they are unaware that other components even exist. Queues facilitate such loosely coupled systems. Components (containers) listen to messages from the queue, respond to messages, perform their jobs, and post messages to the queue about progress, completion status, and errors.

Queues have many benefits:

1. Easy to add processing capacity without coordination; just add more containers that listen to the queue
2. Easy to keep track of overall load by queue depth
3. Easy to have multiple versions of components running side by side by versioning messages and/or topics

4. Easy to implement load balancing as well as redundancy by having multiple consumers process requests in different modes

The downsides of queues are the following:

1. Need to make sure that the queue provides appropriate durability and high availability so it doesn't become a critical SPOF (single point of failure)
2. Containers need to work with the async queue API (could be abstracted away)
3. Implementing request-response requires the somewhat cumbersome listening on response queues

Overall, queues are an excellent mechanism for large-scale systems and they can be utilized in large k8s clusters to ease coordination.

5.4.5.10 Loosely coupled connectivity with data stores

—— This section is a stub for this semester. ——

Another loosely coupled method is to use a data store (for example, Redis) to store messages and then other containers can read them. While possible, this is not the design objective of data stores and the result is often cumbersome, fragile, and doesn't have the best performance. Data stores are optimized for data storage and not for communication. That being said, data stores can be used in conjunction with queues, where a component stores some data in a data store and then sends a message to the queue that data is ready for processing. Multiple components listen to the message and all start processing the data in parallel.

5.4.5.11 k8s ingress

—— This section is a stub for this semester. ——

k8s offers an ingress resource and controller that is designed to expose k8s services to the outside world. You can do it yourself, of course, but many tasks involved in defining ingress are common across most applications for a particular type of ingress, such as a web application, CDN, or DDoS protector. You can also write your own ingress objects.

The ingress object is often used for smart load balancing and TLS termination. Instead of configuring and deploying your own NGINX server, you can benefit from the built-in ingress.

5.4.6 HPA autoscaling in Kubernetes

Autoscaling is a great example of the fundamental notion of a feedback loop we have presented in Section 3.1.9. The feedback in k8s is twofold: (a) the health status of the resources (e.g., pods, deployments, nodes), and, (b) the performance metrics collected from the resources. By default, k8s measures the utilization of the CPU and memory in a pod.

The Horizontal Pod Autoscaling (HPA) operations are described nicely in [6]. Another great description is in Chapter 15 of [11]. The concepts of resource requests and limits require some practice; reference [7] provides great explanations. Reference [8] is also needed for a thorough understanding of the concept of VCPUs.

At the heart of the autoscaling logic is Equation 5.1; it determines how the feedback on performance metrics is processed, in order to determine the new allocation of the resources.

$$C_{k+1} = \left\lceil C_k \cdot \frac{CM_k}{DM} \right\rceil \tag{5.1}$$

Equation 5.1 is computed at (periodic) time instants t_k. At the k-th such instant,

- C_k is the current number of replicas of the scaled resource;
- CM_k is the measured resource metric over the current time interval $[t_{k-1}, t_k)$;
- DM is the desired value of the metric, and,
- C_{k+1} is the calculated number of replicas to be in effect for the next time interval $[t_k, t_{k+1})$.

The ceiling function rounds up the calculated number to the next integer value.

The intuition behind the algorithm in Equation 5.1 is the following. Suppose that the goal is to keep average CPU utilization below a target value DM. CM_k then reports the average CPU utilization over the time interval $[t_{k-1}, t_k)$. The ratio CM_k/DM is an indicator of how well the desired goal is achieved. When the ratio is less than 1, the goal is met; the number of replicas can, therefore, be reduced. When the ratio is greater than 1, the goal is not met; the number of replicas can, therefore, be increased.

5.4.6.1 Metrics used in HPA

HPA can use three types of metrics in making its decisions: *native, custom, and external*. Of interest to us will be only metrics of the native type: these are metrics k8s calculates automatically.

Common native metrics include *average CPU and memory utilization*. These averages are calculated as follows. First, a given pod reports its CPU and memory utilizations as *time averages* over the measurement window. Then an overall average is calculated as *a percentage of the equivalent resource request* on the containers in each Pod. Note that this percentage can exceed 100%, since a pod can consume more than its request, if there is unused resource available.

5.4.7 Load Balancing in Kubernetes

There are several ways that a k8s environment offers load balancing across the pods in a cluster. The default is through kube-proxy. The default kube-proxy mode for

rule-based IP management is iptables, and the iptables mode native method for load distribution is *random selection* [9].

An alternative is through an ingress controller. We quote from [9].

> There are several cases when you might access services using the k8s proxy:
> • Allowing internal traffic
> • Connecting directly to them directly from a computer
> • Debugging services
> • Displaying internal dashboards
>
> However, you should not use this method for production services or to expose your service to the internet. This is because the kube proxy requires you to run kubectl as an authenticated user. In any case, for true load balancing, Ingress offers the most popular method. Ingress operates using a controller with an Ingress resource and a daemon. The Ingress resource is a set of rules governing traffic. The daemon applies the rules inside a specialized k8s pod. The Ingress controller has its own sophisticated capabilities and built-in features for load balancing and can be customized for specific vendors or systems.
>
> A cloud service-based k8s external load balancer may serve as an alternative to Ingress, although the capabilities of these tools are typically provider-dependent. External network load balancers may also lack granular access at the pod level.
>
> There are many varieties of Ingress controllers, with various features, and a range of plugins for Ingress controllers, such as cert-managers that provision SSL certificates automatically.

The load balancing algorithm is configurable. There are several choices. We quote from [9].

> **How to Configure Load Balancer in k8s?** Load balancing, a critical strategy for maximizing availability and scalability, is the process of distributing network traffic efficiently among multiple backend services. A number of k8s load balancer strategies and algorithms for managing external traffic to pods exist. Each has its strengths and weaknesses.
>
> **Round Robin** In a round robin method, a sequence of eligible servers receive new connections in order. This algorithm is static, meaning it does not account for varying speeds or performance issues of individual servers, so a slow server and a better performing server will still receive an equal number of connections. For this reason, round robin load balancing is not always ideal for production traffic and is better for basic load testing.
>
> **Kube-proxy L4 Round Robin Load Balancing** The most basic default k8s load balancing strategy in a typical k8s cluster comes from the kube-proxy. The kube-proxy fields all requests that are sent to the k8s service and routes them. However, because the kube-proxy is actually a process rather than a proxy, it uses iptables rules to implement a virtual IP for the service, adding architecture and complexity to the routing. With each request, additional latency is introduced, and this problem grows with the number of services.
>
> **L7 Round Robin Load Balancing** In most cases, it is essential to route traffic directly to k8s pods and bypass the kube-proxy altogether. Achieve this with an API Gateway for k8s that uses a L7 proxy to manage requests among available k8s pods.
>
> The load balancer tracks the availability of pods with the k8s Endpoints API. When it receives a request for a specific k8s service, the k8s load balancer sorts in order or round robins the request among relevant k8s pods for the service.
>
> **Consistent Hashing/Ring Hash** In consistent hashing algorithms, the k8s load balancer distributes new connections across the servers using a hash that is based on a specified key. Best for load balancing large numbers of cache servers with dynamic content, this algorithm inherently combines load balancing and persistence.
>
> This algorithm is consistent because there is no need to recalculate the entire hash table each time a server is added or removed. Visualizing a circle or ring of nine servers in a pool or cache, adding a tenth server does not force a re-cache of all content. Instead, based on

the outcome of the hash the nine servers already there send about 1/9, an even proportion, of their hits to the new server. Other connections are not disrupted.

The consistent or ring hash approach is used for sticky sessions in which the system ensures the same pod receives all requests from one client by setting a cookie. This method is also used for session affinity, which requires client IP address or some other piece of client state.

The consistent hashing approach is useful for shopping cart applications and other services that maintain per-client state. The need to synchronize states across pods is eliminated when the same clients are routed to the same pods. The likelihood of cache hit also increases as client data is cached on a given pod.

The weakness of ring hash is that client workloads may not be equal, so evenly distributing load between different backend servers can be more challenging. Furthermore, particularly at scale, the hash computation cost can add some latency to requests.

Google's Maglev is a type of consistent hashing algorithm. Maglev has a costly downside for microservices, however: it is expensive to generate the lookup table when a node fails.

Fastest Response Also called weighted response time, the fastest response method sends new connections to whichever server is currently offering the quickest response to new requests or connections. Fastest response is usually measured as time to first byte.

This method works well when the servers in the pool are processing short-lived connections or they contain varying capabilities. HTTP 404 errors are generally the sign of a server having problems, such as a lost connection to a data store. Frequent health checks can help mitigate against this kind of issue.

Fewest Servers A fewest servers strategy determines the fewest number of servers required to satisfy current client load rather than distributing all requests across all servers. Servers deemed excess can either be powered down or de-provisioned, temporarily.

This kind of algorithm works by monitoring changes in response latency as the load adjusts based on server capacity. The k8s load balancer sends connections to the first server in the pool until it is at capacity, and then sends new connections to the next available server. This algorithm is ideal where virtual machines incur a cost, such as in hosted environments.

Least Connections The least connections dynamic k8s load balancing algorithm distributes client requests to the application server with the least number of active connections at time of request. This algorithm takes active connection load into consideration, since an application server may be overloaded due to longer lived connections when application servers have similar specifications.

The weighted least connection algorithm builds on the least connection method. The administrator assigns a weight to each application server to account for their differing characteristics based on various criteria that demonstrate traffic-handling capability. The least connections algorithm is generally adaptive to slower or unhealthy servers, yet offers equal distribution when all servers are healthy. This algorithm works well for both quick and long lived connections.

Resource Based/Least Load Resource based or least load algorithms send new connections to the server with the lightest load, irrespective of the number of connections it has. For example, a load balancer receives one HTTP request requiring a 200-kB response and a second request that requires a 1-kB response. It sends the first to one server and the second to another. For new requests, the server then estimates based on old response times which server is more available—the one still streaming 200 kB or the one sending the 1-kB response to ensure a quick, small request does not get queued behind a long one. However, for non-HTTP traffic, this algorithm will default to the least connections method because the least load is HTTP-specific.

5.5 Ansible

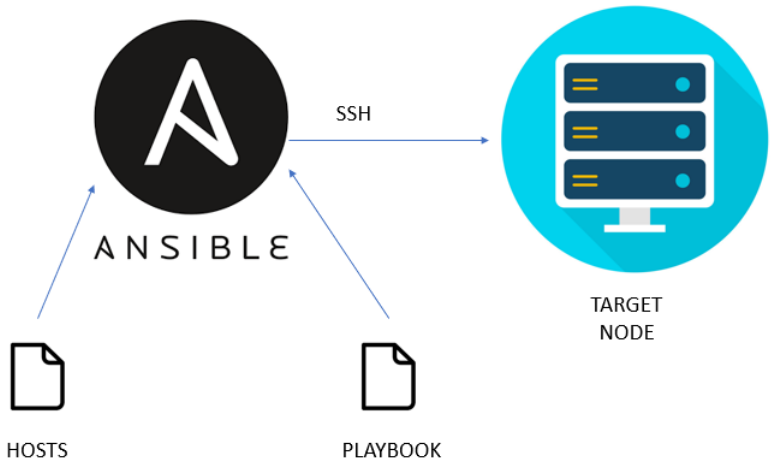


Fig. 5.12: Ansible.

We quote from [21]:

Ansible is an open source IT automation engine that automates provisioning, configuration management, application deployment, orchestration, and many other IT processes.

Use Ansible automation to install software, automate daily tasks, provision infrastructure, improve security and compliance, patch systems, and share automation across your organization.

How does Ansible work? Ansible works by connecting to your nodes and pushing out small programs, called modules to them. Modules are used to accomplish automation tasks in Ansible.

These programs are written to be resource models of the desired state of the system. Ansible then executes these modules and removes them when finished.

Without modules, you'd have to rely on ad-hoc commands and scripting to accomplish tasks.

Ansible is agentless, which means the nodes it manages do not require any software to be installed on them.

Ansible reads information about which machines you want to manage from your inventory. Ansible has a default inventory file, but you can create your own and define which servers you want Ansible to manage.

Ansible uses SSH protocol to connect to servers and run tasks. By default, Ansible uses SSH keys with ssh-agent and connects to remote machines using your current user name. Root logins are not required. You can log in as any user, and then su or sudo to any user.

Once it has connected, Ansible transfers the modules required by your command or playbook to the remote machine(s) for execution.

Ansible uses human-readable YAML templates so users can program repetitive tasks to happen automatically without having to learn an advanced programming language.

Ansible contains built-in modules that you can use to automate tasks, or you can write your own. Ansible modules can be written in any language that can return JSON, such as Ruby, Python, or bash. Windows automation modules are even written in Powershell.

Ansible Playbooks Ansible Playbooks are used to orchestrate IT processes. A playbook is a YAML file containing 1 or more plays, and is used to define the desired state of a system. This differs from an Ansible module, which is a standalone script that can be used inside an Ansible Playbook.

Plays consist of an ordered set of tasks to execute against host selections from your Ansible inventory file. Tasks are the pieces that make up a play, and call Ansible modules. In a play, tasks are executed in the order in which they are written.

When Ansible runs, it is able to keep track of the state of the system. If Ansible scans a system and finds the playbook description of a system and the actual system state don't agree, then Ansible will make whatever changes are necessary for the system to match the playbook.

Ansible includes a "check mode" which allows you to validate playbooks and ad-hoc commands before making any state changes on a system. This shows you what Ansible would do, without actually making any changes.

Handlers in Ansible are used to run a specific task only after a change has been made to the system. They are triggered by tasks and run once, at the end of all of the other plays in the playbook.

Variables are a concept in Ansible that enable you to alter how playbooks run. Variables are used to account for differences between systems, such as package versions or file paths. With Ansible, you can execute playbooks across different systems.

Ansible variables should be defined in relation to what your playbook is actually doing.

Variables follow variable precedence, which defines the order in which variables will override each other. It's important to understand this when including variables in your playbook.

When working with Ansible you will also need to understand collections. Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.

Ansible roles are a special kind of playbook that is fully self-contained and portable with the tasks, variables, configuration templates, and other supporting files that are needed to complete a complex orchestration.

Multiple roles can exist inside a collection allowing easy sharing of content via Automation Hub and Ansible Galaxy.

Orchestration with Ansible In general, automation refers to automating a single task. This is different from orchestration, which is how you can automate a process or workflow that involves many steps across multiple disparate systems.

Cloud orchestration can be used to provision or deploy servers, assign storage capacity, create virtual machines, and manage networking, among other tasks. There are many different orchestration tools that can help you with cloud orchestration. Ansible is one option.

Server configuration and management and application deployments can also be orchestrated with a tool like Ansible.

With application deployments you've got frontend and backend services, databases, monitoring, networks, and storage each with their own role to play and their own configuration and deployment. Orchestration ensures that each step happens the way you need it to.

Ansible enables orchestration by executing the tasks in your playbook in the order in which they are written, so you know that your application deployment processes will happen in the correct order.

5.6 Openstack

—— This section is a stub for this semester. ——

The Wikipedia article in [22] is a good starting point; it also provides a nice history of how this Open Source tool evolved since its inception in July 2010.

We quote from [23].

Understanding OpenStack

Published October 3, 2021

OpenStack® gives you a modular cloud infrastructure that runs off of standard hardware—letting you deploy the tools you need, when you need them, all from one place.

What is OpenStack? OpenStack is an open source platform that uses pooled virtual resources to build and manage private and public clouds. The tools that comprise the OpenStack platform, called "projects," handle the core cloud-computing services of compute, networking, storage, identity, and image services. More than a dozen optional projects can also be bundled together to create unique, deployable clouds.

In virtualization, resources such as storage, CPU, and RAM are abstracted from a variety of vendor-specific programs and split by a hypervisor before being distributed as needed. OpenStack uses a consistent set of application programming interfaces (APIs) to abstract those virtual resources 1 step further into discrete pools used to power standard cloud computing tools that administrators and users interact with directly.

Is OpenStack just a virtualization management platform? Not quite. There are a lot of similarities, but they're not the same.

Yes, OpenStack and virtualization management platforms both sit on top of virtualized resources and can discover, report, and automate processes in vendor-disparate environments.

But while virtualization management platforms make it easier to manipulate the features and functions of virtual resources, OpenStack actually uses the virtual resources to run a combination of tools. These tools create a cloud environment that meets the National Institute of Standards and Technology's 5 criteria of cloud computing: a network, pooled resources, a user interface, provisioning capabilities, and automatic resource control/allocation.

How does OpenStack work? OpenStack is essentially a series of commands known as scripts. Those scripts are bundled into packages called projects that relay tasks that create cloud environments. In order to create those environments, OpenStack relies on 2 other types of software:

– Virtualization that creates a layer of virtual resources abstracted from hardware
– A base operating system (OS) that carries out commands given by OpenStack scripts

Think about it like this: OpenStack itself doesn't virtualize resources, but rather uses them to build clouds. OpenStack also doesn't execute commands, but rather relays them to the base OS. All 3 technologies—OpenStack, virtualization, and the base OS—must work together. That interdependency is why so many OpenStack clouds are deployed using Linux®, which was the inspiration behind RackSpace and NASA's decision to release OpenStack as open source software.

The OpenStack components OpenStack's architecture is made up of numerous open source projects. These projects are used to set up OpenStack's undercloud and overcloud—used by sys admins and cloud users, respectively. Underclouds contain the core components sys admins need to set up and manage end users' OpenStack environments, known as overclouds.

There are 6 stable, core services that handle compute, networking, storage, identity, and images while more than a dozen optional ones vary in developmental maturity. Those 6 core

services are the infrastructure that allows the rest of the projects to handle dashboarding, orchestration, bare-metal provisioning, messaging, containers, and governance.

1. **Nova** Nova is a full management and access tool to OpenStack compute resources—handling scheduling, creation, and deletion.
2. **Neutron** Neutron connects the networks across other OpenStack services.
3. **Swift** Swift is a highly fault-tolerant object storage service that stores and retrieves unstructured data objects using a RESTful API.
4. **Cinder** Cinder provides persistent block storage accessible through a self-service API.
5. **Keystone** Keystone authenticates and authorizes all OpenStack services. It's also the endpoint catalog for all services.
6. **Glance** Glance stores and retrieves virtual machine disk images from a variety of locations.

What can I do with OpenStack?

Private clouds Private cloud distributions run on OpenStack can provide more substantial benefits than private clouds built using custom code. IDC evaluated the value of Red Hat OpenStack Platform for private clouds and found that organizations realized annual benefits of $6.81 million.

Network functions virtualization 451 Research found that using OpenStack for network functions virtualization (NFV)—which involves separating a network's key functions so they can be distributed among environments—could very well be the next big thing. It's on the agenda of virtually every global communications services provider surveyed by the analyst.

Public clouds OpenStack is the leading open source option for building public cloud environments. Whether your company is a multibillion-dollar publicly traded enterprise or a startup, you can use OpenStack to set up public clouds with services that compete with major public cloud providers.

Containers OpenStack is a stable foundation for public and private clouds. Containers speed up application delivery while simplifying application deployment and management. Running containers on OpenStack can scale containers' benefits from single, siloed teams to enterprise-wide interdepartmental operations.

Why Red Hat OpenStack? Because we stabilize OpenStack for enterprises while staying true to its open source roots. We keep OpenStack open source, which gives you complete control over the cloud infrastructure and everything that relies on it. You can modify a Red Hat® OpenStack deployment to work with (or without) any vendor because the code isn't locked behind proprietary walls—it's yours. And we work closely with cloud providers so you get peak performance no matter what infrastructure you deploy on.

But it's not just about products, support, consulting, and training. We're your partner—helping you introduce an open source culture to your enterprise.

5.7 Problem Section

Problems on deployments

Problem 5.1. Deployments. Assume you are running an online service like Uber, Netflix, Facebook etc. Which deployment model would you use for the following services and why?

- Search Microservice that enables users to search for cars, movies or friends.
- Recommendation microservice (note that services like Netflix have a default recommendation when the recommendation microservice is not functioning).
- Movie encoding Microservice (a microservice that is not on the "streaming path").

Problem 5.2. Deployments. The following are some of the benefits of canary deployments. Please provide further context on how canary deployments enable the following:

- A/B testing: ...
- Capacity Test: ...
- User Feedback: ...
- No cold-starts: ....
- No Downtime ...
- Easy Rollback ...

Your answers should be minimally 2 sentences and not more than 3-4 sentences.

Problems on k8s

Problem 5.3. k8s, top-level business requirements. Read Reference [16], The State of Kubernetes 2023. It contains a report by VMware on the business adoption of k8s in the years 2022 and 2023.

1. In your opinion, what business requirements are described in the 2022 report?
2. In the 2023 report?
3. Translate one business requirement into concrete technical requirements.
4. Expand on the "tool selection" challenge.

Problem 5.4. k8s concepts and terminology. Pod is a fundamental concept in k8s. The word pod as well as the acronym POD are used in several contexts.

1. Find its precise, scientific meaning in Biology, Geology and Computing.
2. Repeat for its use in the Angling, Aviation and Cooking professions.
3. Expand the acronym POD in the fields of financing, sales, shipping, business, sales, and marketing.
4. (Optional) We may use tri*pods* for taking photographs; there are *pod*iatrists in the medical profession. How is the word used in these contexts?
5. (Optional) We listen to *pod*casts. How is the word used in these contexts?

6. (Last but not least) Search for synonyms of pod. Which one, in your opinion, *comes closest* to the use of the word in k8s? Why?

Problem 5.5. k8s concepts and terminology. Consider an "e-commerce store" application A that has been packaged into the following containers C1, C2 and C3:

- C1: front end. It interfaces with customers accessing the app through the internet. It accepts all inputs from the customers and returns to them all outputs.
- C2: main logic. This part processes all orders from customers. In a nutshell, it checks and updates the inventory and handles the credit card payments.
- C3: database handling. This part maintains three separate databases: (a) inventory, (b) payments, and, (c) logs.
- If you feel you need additional logic in the application, feel free to add it in one or more separate containers.

Suppose you are using k8s to deploy and manage this application. Read Section 5.4.2.

1. How many pods would you define? Why?
2. Repeat for replicas.
3. Repeat for deployments.
4. Define a couple of objects.
5. Define a couple of controllers.
6. Define a spec for your application.
7. Describe a potential state for your deployment. Mention a few reasons that will result in this state.
8. Supply IP addresses for your application. Justify your choice of private/public addresses.
9. What is the minimum number of addresses you need to specify? Why?
10. What is the maximum number of addresses you need to specify? Why?
11. How many services would you define? Why?
12. Who will access these services? Why?

Problem 5.6. k8s concepts and terminology. Consider the following YAML file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 5
  strategy:
```

```
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
       - name: nginx
         image: nginx
         resources:
           limits:
             memory: 200Mi
           requests:
             cpu: 100m
             memory: 200Mi
         ports:
          - containerPort: 80
```

1. What kind of k8s object does it define?
2. What is the name of the application is it running?
3. What are the requirements of the application?
4. How is the application accessed by its users?
5. Consider a worker node with a CPU capacity of X and memory size of Y units. How many copies of this application can be scheduled on this node, as a function of X and Y? State your assumptions clearly.

Problem 5.7. How to put applications into pods? Consider the placement of an application into one or more pods.

Albert Steinein, a self-proclaimed software expert, suggests the following set of "best practices".

- BP1: if the application has $N \cdot 100K$ lines of code, put it in N pods.
- BP2: if the application calls M functions, put each function in its own pod. Put the application in an additional pod.
- If the application accesses L databases, put each database in its own pod. Put the application in an additional pod.

1. Comment on each one of the suggestions.
2. Make your own suggestion. Justify it by supplying an example application for which the suggestion "works".
3. Consider the DNS application we mentioned in Problem 2.9, page 36. Suggest and justify a placement into pods. State your assumptions clearly.

Problem 5.8. k8s Deployments. Read the described use cases for deployments in the k8s documentation, see `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`. Choose a use case.

1. Describe the use case in your own words.

2. State the advantages k8s offers in this use case.
3. Run a scenario of the use case. Supply the yaml files you used.

Problem 5.9. k8s metrics. Consider a metric, e.g., CPU utilization. The value of this metric is a random quantity, affected by several factors. k8s calculates an average value, call it U, over a time interval T.

1. Find the default value of this time interval.
2. The number N of pods may change from interval to interval. Find out how k8s uses N in its calculation of U.
3. (Optional) The average CPU utilization over a longer time interval (say one month) is of interest. Can you relate U to this average?

Problem 5.10. k8s best practices. Find out any reference on k8s best practices. If your search came out empty, try `https://www.densify.com/kubernetes-tools/kubernetes-best-practices/`. It describes ten such practices.

1. Which role(s) are these suggestions targeting?
2. Pick one suggested practice and expand on its pros and cons.
3. Find out the default value of the "liveness probe" frequency.
4. Find out the default values of the "Resource Requests and Limits" parameters.

Problem 5.11. k8s HPA. Read the HPA walkthrough described in `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/`.

In order to run the experiments, you must select one of the three environments mentioned: minikube, killercoda or Play with Kubernetes. If you have access to an application other than the php-apache server, we'd prefer to use it instead.

1. Run the experiments as described. Observe and report how HPA scales up and down.
2. Run experiments with different CPU target percentages, keeping load generation the same. Observe and report how HPA scales up and down. Explain any differences from the results of the previous question.

Problem 5.12. k8s HPA. Consider Equation 5.1, the standard HPA calculation. Suppose, for simplicity, the following:

- initially you have one replica running;
- cool down period t_{cd} is 0;
- the time t_{up} it takes to spawn a new container is 0.

1. Generate 20 different values for the current metric; all of them should be increasing. Plot the number of replicas HPA will come up with.
2. Generate 20 different values for the current metric; all of them should be decreasing. Plot the number of replicas HPA will come up with.
3. Generate 40 randomly decreasing and increasing values for the current metric. Plot the number of replicas HPA will come up with.

Problem 5.13. k8s HPA. Consider Equation 5.1, the standard HPA calculation. Suppose, for simplicity, the following:

- initially you have one replica running;
- cool down period t_{cd} is 0;
- the time t_{up} it takes to spawn a new container is 1 second.

1. Generate 20 different values for the current metric; all of them should be increasing. Plot the number of replicas HPA will come up with.
2. Generate 20 different values for the current metric; all of them should be decreasing. Plot the number of replicas HPA will come up with.
3. Generate 40 randomly decreasing and increasing values for the current metric. Plot the number of replicas HPA will come up with.

Problem 5.14. k8s HPA and the effect of the cool down period. Consider Equation 5.1, the standard HPA calculation. Suppose, for simplicity, the following:

- initially you have one replica running;
- cool down period t_{cd} is 5 minutes;
- the time t_{up} it takes to spawn a new container is 1 second.

Generate 100 random values for the current metric with the following properties. The first 20 are all increasing; the next 40 are all decreasing; the last 40 are randomly decreasing and increasing.

If you need to make further assumptions, state them clearly.

1. Plot the number of replicas HPA will come up with.
2. Discuss how the t_{cd} value affected this plot.
3. Now assume that $t_{cd} = 0$. Plot again the number of replicas, assuming the same 100 random values for the current metric.

Problem 5.15. k8s, the difficulty of configuring HPA. The period T at which measurements are taken from pods and processed is a configurable parameter, with a 15 second default value.

1. List some factors that affect the selection of T.
2. Describe an application and a scenario in which the default value would not work.
3. Describe an application and a scenario in which the default value would work.
4. Suppose that the time to launch a new pod is 500ms. What values for T would not make sense in this case?

Problem 5.16. k8s HPA and the difficulty of using custom metrics. Read Reference [20]. It describes a particular networking "pain" related to custom metrics and their use in HPA.

1. Describe, in your own words, what the problem was.
2. Describe, in your own words, the workaround.

Problem 5.17. k8s, practical issues with the HPA formula. Read Reference [13]. It cites some (perceived) issues with the default scaling formula 5.1, page 127.

1. Design and run an experiment to showcase the issue.

Problem 5.18. Provisioning resources and scaling to match bursty workloads. Consider the bursty DNS workload scenario we discussed in Problem 2.9, page 36. We have k8s managing a number N of DNS server instances that serve this load.

Recall that Figure 2.1 depicts the rate of requests over time. Assume that the vertical axis depicts true scale. Assume that one instance can handle the average load.

1. Describe how you would expect HPA to work in this scenario.
2. What is a reasonable value for N? Why?
3. Choosing a high value for N is good for the objective of absorbing unexpected, highly bursty workloads. On the other hand, such a choice increases the risk of paying a higher $$ cost in case of misconfigurations, attacks, etc. What tradeoff would you make and why?

Problem 5.19. k8s, specific features. Find a tutorial on specific aspects of k8s. You may start with `https://kubernetes.io/docs/tutorials/`, the Linux Foundation `https://training.linuxfoundation.org/resources/tutorials/getting-started-with-kubernetes-is-easy-with-minikube/` or Youtube, among others.

1. Describe the feature in your own words.
2. What BR or TR does the feature relate to? Which hat benefits from it?
3. Summarize your findings.

Problem 5.20. Kubelet features. Find out specific features of the kubelet component of k8s. You may start with `https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/`.

Problem 5.21. Kube-proxy features. Find out specific features of the kube-proxy component of k8s. You may start with `https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/`.

Problem 5.22. k8s and CI/CD. Handout H6.1 mentions some advantages k8s offers in the CI/CD use case.

1. Describe briefly what a CI/CD pipeline is.
2. How many advantages does the handout mention? Expand on two of them, using your own examples. Mention explicitly who benefits from these advantages.
3. Since there is no free lunch in engineering, there must be disadvantages. Identify at least one. Mention explicitly who "suffers" from these disadvantages.

Problem 5.23. Declarative vs imperative APIs. By design, k8s is based on a *declarative*, resource-centric API.

1. Describe what exactly we mean by declarative APIs.
2. Describe what exactly we mean by imperative APIs.
3. Give two examples of both types.
4. List as many advantages of declarative APIs as you can find.

5. Since there is no free lunch in engineering, find as many disadvantages of declarative APIs as you can.
6. Why, in your opinion, declarative APIs are "better"?

Problem 5.24. k8s ecosystem. Check the list of k8s Certified Service Providers in `https://landscape.cncf.io/`, the "CNCF Cloud Native Interactive Landscape" site[1].

Pick one of the providers at random, preferably a startup, by clicking on their tile. Visit their website.

1. Find out what exactly the provider offers. For example, Mirantis says "Mirantis provides full-stack support for clouds built with k8s and related open source software, using a flexible GitOps model for lifecycle management." (Quoted on June 4, 2023.)
2. Report on anything interesting you find there.

Problem 5.25. vCPU calculations. Obtain the spec for any virtualized server from the references in Chapter 2.

1. Find out the number of cores in the server.
2. Determine the number of vCPUs the server can support.

Huh?

Problem 5.26. Automation and Automaton. These two words look related - only a trained eye would catch the missing i.

1. Give an example of automation in everyday life.
2. Give an example of an automaton.
3. Was the chess-playing machine called the Turk an automaton?
4. What advances, if at all possible, in AI would be needed to automate the process of midterm-exam taking? If you find a way, you can certainly move into `https://en.wikipedia.org/wiki/Waddesdon_Manor` and watch The Elephant Automaton in the East Gallery to your heart's content.
   The experienced movers shown in Figure 5.13 will be happy to sail you there.

---

[1] On 2023-06-03T21:12:12, there were 235 providers listed. They are all potential employers.

Fig. 5.13: PV Sailing: we can sail you anywhere in the world...

References

References marked as Handout H5.x are required reading material; you will find them in the required handouts binder in moodle. The rest are used in some form in the text or problems.

1. **Reference R5.1:** Dennis Smith at al, "Market Guide for Cloud Management Tooling", `https://morpheusdata.com/cloud-management-tooling-thank-you/`
2. **Reference R5.2:** What is Cloud Orchestration? `https://www.vmware.com/topics/glossary/content/cloud-orchestration.html`
3. **Reference R5.3:** Terraform, `https://www.ibm.com/cloud/learn/terraform`
4. **Handout H5.1:** Dan Sullivan, The Gorilla Guide to... Getting Started with Kubernetes, ActualTech Media, 2022
5. **Handout H5.2:** What is Kubernetes? `https://www.vmware.com/topics/glossary/content/kubernetes.html`
6. **Reference R5.4:** Horizontal Pod Autoscaling `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`
7. **Reference R5.5:** Kubernetes best practices: Resource requests and limits `https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits`

8. **Reference R5.6:** James Timmerwilke, What is a vCPU and How Do You Calculate vCPU to CPU? https://www.datacenters.com/news/what-is-a-vcpu-and-how-do-you-calculate-vcpu-to-cpu
9. **Reference R5.7:** Kubernetes Load Balancer, https://avinetworks.com/glossary/kubernetes-load-balancer/
10. **Reference R5.8:** Joseph Heck, Kubernetes for Developers, Packt Publishing, 2018
11. **Reference R5.9:** Marko Luksa, Kubernetes in Action, Manning Publications, 2018
12. **Reference R5.10:** Gigi Sayfan, Mastering Kubernetes, 2nd Edition, Packt Publishing, 2018.
13. **Reference R5.11:** The poor state of Kubernetes horizontal pod autoscaling, https://www.wjwh.eu/posts/2020-01-04-kubernetes-autoscaling.html
14. **Reference R5.12:** Daniel Weibel, "How to autoscale apps on Kubernetes with custom metrics", 2019, https://learnk8s.io/autoscaling-apps-kubernetes
15. **Reference R5.13:** Pavan Belagatti, Kubernetes for Everyone, JFrog Ltd.
16. **Reference R5.14:** The State of Kubernetes 2023, Presented by: VMware https://tanzu.vmware.com/content/ebooks/stateofkubernetes-2023
17. **Reference R5.15:** State of Kubernetes 2023, by VMware, https://tanzu.vmware.com/content/ebooks/stateofkubernetes-2023
18. **Reference R5.16:** Vladimir Kaplarevic, "Understanding Kubernetes Architecture with Diagrams", November 12, 2019, https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams
19. **Reference R5.17:** Kubernetes article in Wikipedia, https://en.wikipedia.org/wiki/Kubernetes
20. **Reference R5.18:** Marko Lukša, "Kubernetes autoscaling based on custom metrics without using a host port", Feb 3, 2017. https://medium.com/@marko.luksa/kubernetes-autoscaling-based-on-custom-metrics-without-using-a-host-port-b783ed6241ac
21. **Reference R5.19:** Learning Ansible basics, https://www.redhat.com/en/topics/automation/learning-ansible-tutorial
22. **Reference R5.20:** Openstack https://en.wikipedia.org/wiki/OpenStack
23. **Reference R5.21:** Openstack https://www.redhat.com/en/topics/openstack

Topic 6
Design problems for the cloud architect

Objectives: After reading this chapter, you should be able to: (a) state some of the fundamental design problems the cloud architect is called to solve, (b) describe some solutions to such problems, (c) explain why tradeoffs must be made in proposed solutions, and, (d) evaluate such tradeoffs.

The range of problems the cloud "architect" deals with is quite wide; one person cannot solve them all. In this chapter, we further refine the responsibilities of the cloud "architect" role we presented in Chapter 1 into several subspecialties. Regardless of the subspecialty, the architect must deal with a set of technical requirements and examine options for satisfying them. We stress the importance of understanding the notion of tradeoffs and why they cannot be avoided. With this framework of mind in place, we then describe a few problems that the cloud architect deals with.

6.1 Sub-specialities of the cloud architect roles

6.1.1 The Distinguished cloud architect

Ειναι ολα Ελληνικα για μενα Καλημερα σε ολους

This is the architect we described in Section 1.4.2.1. There is a small number of such architects in major Cloud providers. They might focus within a specific space, for example a Distinguished Cloud Architect in Machine Learning/Artificial Intelligence or a Distinguished Cloud Architect in all Platform areas. In some occasions, Cloud consumers may have such a title but it requires a larger size company such that the coordination across multiple projects necessitates that high level position. Larger Cloud Providers also have dedicated meetings at the Distinguished level to

allow technical information to flow across multiple endeavors. You may also hear similar names like “Fellow”, “Master” architect, etc.

In addition to the responsibilities we mentioned in Section 1.4.2.1, this architect communicates with the C-suite (see Example 4.13, page 85) and provides direction to the subspecialty architects of this section.

We expect you to become one after[1] you pass this course.

6.1.2 The Cloud Solutions Architect

According to cloud provider literature, this role works for the cloud consumer[2]. Responsibilities require multiyear experience. This fact is reflected in industry certifications that come in two levels, Associate and Professional (for more on certifications, see Chapter C in the Appendix). For the Associate role, we quote from `https://aws.amazon.com/certification/certified-solutions-architect-associate/`:

> This credential helps organizations identify and develop talent with critical skills for implementing cloud initiatives. Earning AWS Certified Solutions Architect – Associate validates the ability to design and implement distributed systems on AWS.
>
> AWS Certified Solutions Architect – Associate is intended for anyone with one or more years of hands-on experience designing available, cost-efficient, fault-tolerant, and scalable distributed systems on AWS. Before you take this exam, we recommend you have:
>
> - One year of hands-on experience with AWS technology, including using compute, networking, storage, and database AWS services as well as AWS deployment and management services
> - Experience deploying, managing, and operating workloads on AWS as well as implementing security controls and compliance requirements
> - Familiarity with using both the AWS Management Console and the AWS Command Line Interface (CLI)
> - Understanding of the AWS Well-Architected Framework, AWS networking, security services, and the AWS global infrastructure
> - Ability to identify which AWS services meet a given technical requirement and to define technical requirements for an AWS-based application

For the Professional role, we quote from `https://aws.amazon.com/certification/certified-solutions-architect-professional/`

> This credential helps organizations identify and develop talent with critical skills for implementing cloud initiatives. Earning AWS Certified Solutions Architect – Professional validates the ability to design, deploy, and evaluate applications on AWS within diverse, complex requirements.
>
> AWS Certified Solutions Architect – Professional is intended for individuals with two or more years of hands-on experience designing and deploying cloud architecture on AWS. Before you take this exam, we recommend you have:

---

[1] Well, not soon after, anyway.

[2] We will argue that the provider has a need for a cloud solutions architect as well; some problems are common to both. For example, the load balancing problem we discuss in Section 6.7.

- Familiarity with AWS CLI, AWS APIs, AWS CloudFormation templates, the AWS Billing Console, the AWS Management Console, a scripting language, and Windows and Linux environments
- Ability to provide best practice guidance on the architectural design across multiple applications and projects of the enterprise as well as an ability to map business objectives to application/architecture requirements
- Ability to evaluate cloud application requirements and make architectural recommendations for implementation, deployment, and provisioning applications on AWS
- Ability to design a hybrid architecture using key AWS technologies (e.g., VPN, AWS Direct Connect) as well as a continuous integration and deployment process

6.1.3 The cloud network architect

Up to six networks may be involved in a cloud computing environment. There is an architect role for each network type, of course.

1. The provider's network inside a datacenter; it interconnects physical servers, storage devices and the gateways, as we have briefly described in Section 2.2.1.3.
2. The network that interconnects the datacenters of the cloud provider. This is usually called the Data Center Interconnect (DCI). In Example 6.1, we show the DCIs for the big 3 providers.
3. The network that connects the cloud consumers to the datacenters of the provider. This network can be the public internet or VPNs offered by the providers, as we show in Example 6.2.
4. The network that interconnects the providers' datacenters in the multicloud environment. This is a virtual, software-defined network; it is a fairly new, still emerging technology[3].
5. The network that interconnects the consumer's users (e.g., an enterprise WAN). This network has nothing to do with cloud computing. And last but not least,
6. The network that connects the VMs and/or containers of a consumer together. This network is virtual; a common name for it is Virtual Private Cloud Network (VPCN). We will address issues in designing this network in Section 6.6.

Example 6.1. **The DCIs for the big three providers.** Figures 6.1 and 6.2 depict the DCI networks connecting the datacenters of the "big 3".

△

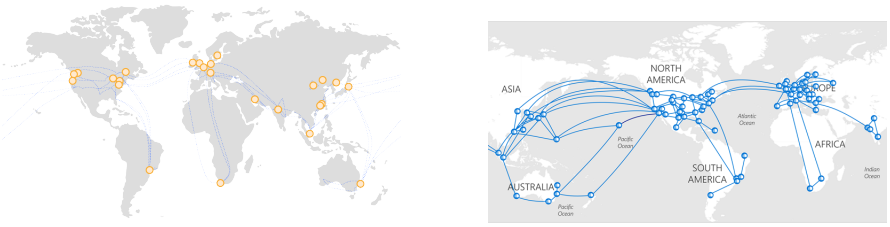
Example 6.2. **VPNs provided by cloud providers.** Many of the major cloud providers even give customers a preconfigured VPN as part of their subscription packages:

- Amazon Web Services offers *Direct Connect*. See [24].
- Microsoft Azure offers *ExpressRoute*. See [25].
- Google Cloud offers *Dedicated Interconnect*. See [26].

  Even OpenStack provides the *OpenStack Public Cloud Passport*: see [27]

△

[3] We"l address it in the advanced course.



(a) Amazon's Global Network. (b) Azure's SWAN network

Fig. 6.1: Amazon's and Azure's DCI networks.

Fig. 6.2: GCP's B4 DCI.

Example 6.3. **VPCN.** (For more details, see `https://cloud.google.com/vpc/docs/vpc`.) Consider a consumer who has a SaaS application that autoscales and a PaaS application that runs on two VMS. The consumer wants all of them on the same VLAN.

The consumer has no control over which physical servers house her/his software.

- This LAN is a virtual network.
- Each "node" on this network has a direct, virtual connection to each other node. This connection is a virtual point-to-point link or "pipe".
- The entire switch fabric then behaves like a layer 2, virtual switch.
- As containers autoscale, IP addresses must be assigned in a dynamic fashion.

△

6.1.4 The Cloud Data Architect

A data architect is an individual who is responsible for designing, creating, deploying and managing an organization's data architecture. Data architects define how the data will be stored, consumed, integrated and managed by different data entities and IT systems, as well as any applications using or processing that data in some way.

Data is growing extremely fast and the role is extremely important as companies are trying to define their data strategy. Technological trends are fragmenting the data landscape. The speed of software delivery is changing with the new methodologies at a cost of increased data complexity. The rapid growth of data and intensive data

consumption make operational systems hard to scale. Lastly, there are privacy, security and regulatory concerns that a data architect must consider when building data platforms.

There are also different forms of data architectures. Data warehouses and data lakes are useful to utilize the data and their capabilities continuously become better. Machine Learning applications have also becoming more data intensive with higher requirements for responsiveness. The Cloud Data Architect must design system across multiple stakeholders that are producing or consuming the data. Some of them are deep technical like backend engineers, machine learning scientists, data engineers and some of them might be no code stakeholders.

> Cloud data architectures contain the rules, models, and policies that define how data is collected, stored, used and managed in the cloud within a business or an organization. Cloud data architectures manage the flow of data and how that data is processed and distributed across stakeholders and other applications for reporting, analytics, and other uses.

And what would be required skills for a data architect? We quote a (rather arbitrary, Top 10) list from [29]:

> **Top 10 Skills for Data Architects**
>
> **Business Skill** Data Architects should have vast knowledge of the industry, trends and competition so they can solve issues efficiently. Ability to provide data for business purpose to increase customer satisfaction, identification of key performance indicators, product improvement, business decisions on growth, diversification of services, expenditure evaluation and risk assessment evaluation.
>
> **Programming Skills like SQL, Python and Java** *SQL (Structured Query Language)* is a must have skill for Data Architects. SQL is used to query and manipulate data from relational database models. We have different types of SQL statements such as Data Manipulation Language (DML), Data Definition Language (DDL), Data Control Language (DCL) and Transactional Control Language (TCL).
>
> DML: Select, Insert, Update, Delete
> DDL: Create, Alter, Drop
> DCL: Grant, Revoke
> TCL: Commit, Rollback, Save Transaction
>
> *Python* as a multi-purpose language can be used to develop several applications used in wide range for data analysis.
>
> *Java* is a class-based, object-oriented, general programming language used to build dynamic applications used on Mobile, desktop, Web, and Server side.
>
> **Data Modelling** Data modelling is an important skill. It shows the Architect understands the data model and design principles which enhance business processes. Understanding schema, entities, data flow, hierarchy are attributes of communication between Architects and data in their organization.
>
> **Applied Math's and Statistics** Math's and statistics are skills needed for Data Architect positions due to analytical reasoning, computation, interpretation and presentation of data to solve business problems thereby enhancing decision making processes.
>
> **Design Skills** Data Architects should have the ability to design models that bring solution for businesses and services for users. Data should be easily available to authorized users, and data elements should be well defined so that they are correctly interpreted. Data quality and integrity should be managed by various design techniques.
>
> **Machine Learning and Natural Language Processing** Knowledge of Machine Learning, Pattern Recognition and Natural Language Processing is very important because Data Architects understand interaction between computers and human language to resolve data driven problems. They also use clusters for handling data and text mining.

Excellent Communication Skill Data Architects should be able to communicate and influence their customers positively.

Databases and Cloud Architecture Data Architects most part of the job is on the Cloud Infrastructure (Iaas, Paas, Saas). Understanding Cloud technologies such as Microsoft Azure, Google Cloud, Amazon Web Services (AWS) and Oracle Cloud is very important. Also, knowledge of NoSQL database is required as well as experience to handle data technologies thatsuch as; MongoDB, Hadoop, Cassandra and Pig, MapReduce and HBase.

Ability to use a variety of Design/Visualization tools A good Architect is not limited to one toolset. A lot of design and visualization tools out there continue evolving and new ones possibly will be introduced in the future.

Creative and Analytical Problem Solving This skill is amongst the top 10 skills for Data Architects. Becoming creative as well as analytical in resolving issues is a sought-after skill for Architect positions.

In Conclusion, the above listed skills if acquired, will make a great Data Architect. Remember these skills take some years to acquire, so have a set objective, keep learning and training to become an outstanding Data Architect.

6.1.5 The cloud storage architect

—— This section is a stub for this semester. ——

6.1.6 The cloud software architect

—— This section is a stub for this semester. ——

6.1.7 The cloud security architect

—— This section is a stub for this semester. ——

6.1.8 The cloud tooling architect

—— This section is a stub for this semester. ——

6.1.9 The cloud financial architect

—— This section is a stub for this semester. ——

6.1.10 Other architects

There are other architects involved in the cloud computing ecosystem. In Figure 6.3, we show some who beautify the ecosystem.

(a) The architects... (b) ...and their tools

Fig. 6.3: (Datacenter) landscape architects and their tools.

6.2 The generic technical requirements

The most common, high-level technical requirements imposed on a cloud computing solution as a result of addressing given business requirements relate to the following concepts:[4]

1. Elasticity
2. Availability
3. Observability
4. Manageability
5. Performance
6. Security

Each one can be expressed as several, more refined technical requirements, as we'll see in more detail later in this chapter; you will have ample opportunity to further refine them in your project. There are several KPIs and metrics associated with each one as well.

There are questions that are common to all such requirements. The silliest question a budding architect can ask is: what "knobs" are at my disposal when I try to address a specific technical requirement? Keep this one in mind, when studying any problem in Sections 6.4 through 6.13. Close to this one in level of silliness

[4] We have harped on this point in Section 4.1.1: to be more accurate, we should always express a requirement with an actionable verb, e.g., "Provide high availability".

are: do the knobs depend on the type of -aaS? do they depend on whether I work for a provider or consumer? Having answers to all three will greatly speed up the transformation from a bud to a flower - no pun intended.

6.2.1 Elasticity

"The cloud system should be elastic." We would argue that the ability of the cloud to provide (properly priced) elasticity is one of the two main benefits of cloud computing for the consumer - the other being the reduction in management headaches.

We quote from [28]:

What is Cloud Elasticity?

Cloud Elasticity is the property of a cloud to grow or shrink capacity for CPU, memory, and storage resources to adapt to the changing demands of an organization. Cloud Elasticity can be automatic, without need to perform capacity planning in advance of the occasion, or it can be a manual process where the organization is notified they are running low on resources and can then decide to add or reduce capacity when needed. Monitoring tools offered by the cloud provider dynamically adjust the resources allocated to an organization without impacting existing cloud-based operations.

A cloud provider is said to have more or less elasticity depending on the degree to which it is able to adapt to workload changes by provisioning or deprovisioning resources autonomously to match demand as closely as possible. This eliminates the need for IT administration staff to monitor resources to determine if additional CPU, memory, or storage resources are needed, or whether excess capacity can be decommissioned. Cloud Elasticity is often associated with horizontal scaling (scale-out) architecture, and it generally associated with public cloud provider resources that are billed on a pay-as-you-go basis. This approach brings real-time cloud expenditures more closely in alignment with the actual consumption of cloud services, for example when virtual machines (VMs) are spun up or down as demand for a particular application or service varies over time. Cloud Elasticity provides businesses and IT organizations the ability to meet any unexpected jump in demand, without the need to maintain standby equipment to handle that demand. An organization that normally runs certain processes on-premises can 'cloudburst' to take advantage of Cloud Elasticity and meet that demand, returning to on-premises operations only when the demand has passed. Thus, the result of cloud elasticity is savings in infrastructure costs, in human capital, and in overall IT costs.

Why is Cloud Elasticity Important?

Without Cloud Elasticity, organizations would have to pay for capacity that remained unused for most of the time, as well as manage and maintain that capacity with OS upgrades, patches, and component failures. It is Cloud Elasticity that in many ways defines cloud computing and differentiates it from other computing models such as client-server, grid computing, or legacy infrastructure. Cloud Elasticity helps businesses avoid either over-provisioning (deploying and allocating more IT resources than needed to serve current demands) or under-provisioning (not allocating enough IT resources to meet existing or imminent demand). Organizations that over-provision spend more than is necessary to meet their needs, wasting valuable capital which could be applied elsewhere. Even if an organization is already utilizing public cloud, without elasticity, thousands of dollars could be wasted on unused VMs every year. Under-provisioning can lead to the inability to serve existing demand, which could lead to unacceptable latency, user dissatisfaction, and ultimately loss of business as customers abandon long online and take their business to more

responsive organizations. In this way the lack of Cloud Elasticity can lead to lost business and severe bottom-line impacts.

How does Cloud Elasticity Work? Cloud Elasticity enables organizations to rapidly scale capacity up or down, either automatically or manually. Cloud Elasticity can refer to 'cloudbursting' from on-premises infrastructure into the public cloud for example to meet a sudden or seasonal demand. Cloud Elasticity can also refer to the ability to grow or shrink the resources used by a cloud-based application. Cloud Elasticity can be triggered and executed automatically based on workload trends, or can be manually instantiated, often in minutes. Before organizations had the ability to leverage Cloud Elasticity, they would have to either have additional stand-by capacity already on hand or would need to order, configure, and install additional capacity, a process which could take weeks or months. If and when demand eases, capacity can be removed in minutes. In this manner organizations pay only for the amount of resources in use at any given time, without the need to acquire or retire on-premises infrastructure to meet elastic demand. Typical use cases for Cloud Elasticity include:

• Retail or e-tail holiday seasonal demand, in which demand increases dramatically from Black Friday shopping specials until the end of the holiday season in early January

• School district registration which spikes in demand during the spring and wanes after the school term begins

• Businesses that see a sudden spike in demand due to a popular product introduction or social media boost, such as a streaming service like Netflix adding VMs and storage to meet demand for a new release or positive review.

• Disaster Recovery and Business Continuity (DR/BC). Organizations can leverage public cloud capabilities to provide off-site snapshots or backups of critical data and applications, and spin up VMs in the cloud if on-premises infrastructure suffers an outage or loss.

• Scale virtual desktop infrastructure in the cloud for temporary workers or contractors or for applications such as remote learning

• Scale infrastructure into the cloud for test and development activities and tear it down once test/dev work is complete.

• Unplanned projects with short timelines

• Temporary projects like data analytics, batch processing, media rendering, etc.

What are the Benefits of Cloud Elasticity?

The benefits of cloud elasticity include:

Agility: By eliminating the need to purchase, configure, and install new infrastructure when demand changes, Cloud Elasticity prevents the need to plan for such unexpected demand spikes, and enables organizations to meet any unexpected demand, whether due to seasonal spike, mention on Reddit, or selection by Oprah's book club.

Pay-as-needed pricing: Rather than paying for infrastructure whether or not is is being used, Cloud Elasticity enables organizations to pay only for the resources that are in use at any given point tin time, closely tracking IT expenditures to the actual demand in real-time. In this way, although spending may fluctuate, organizations can 'right-size' their infrastructure as elasticity automatically allocates or deallocates resources on the basis of real-time demand. Amazon has stated that organizations that adopt its instance scheduler with their EC2 cloud service can achieve savings of over 60 percent versus organizations that do not.

High Availability: Cloud elasticity facilitates both high availability and fault tolerance, since VMs or containers can be replicated if they appear to be failing, helping to ensure that business services are uninterrupted and that users do not experience downtime. This helps ensure that users perceive a consistent and predictable experience, even as resources are provisioned or deprovisioned automatically and without impact on operations.

Efficiency: As with most automations, the ability to autonomously adjust cloud resources as needed enables IT staff to shift their focus away from provisioning and onto projects that are more beneficial to the organization.

Speed/Time-to-market: organizations have access to capacity in minutes instead of the weeks or months it may take through a traditional procurement process.

What are the Challenges in Cloud Elasticity?

Cloud Elasticity is only useful to organizations that experience rapid or periodic increases or decreases in demand for IT services. Organizations with predictable, steady demand most likely would not find an advantage in the benefits of Cloud Elasticity. Here are some potential challenges with Cloud Elasticity

Time to provision: Although cloud VMs can be spun up on-demand, there can still be a lag time of up to several minutes before it is available for use. This may or not be enough time base on a specific application or service demands, and can impact performance when a sudden surge occurs, such as when a sign-on storm occurs at the beginning of the business day.

Cloud Provider Lock-in: Although all major public cloud providers offer Cloud Elasticity solution, each are implemented differently, which cloud mean that organizations are locked into a single vendor for their cloud needs.

Security Impact: Cloud services that spin up and down in an elastic fashion can impact existing security workflows and require them to be reimagined. Since elastic systems are ephemeral, incident response may be impacted, for example when a server experiencing a security issue spins down as demand wanes.

Resource Availability: Cloud Elasticity does require modifications to existing cloud or on-premises deployments. Organizations that do not outsource their IT management will need to acquire technical expertise including architects, developers, and admins to help ensure that a Cloud Elasticity plan is properly configured to meet the organization's specific needs. This can also introduce a learning curve delay as the newly acquired talent come up to speed on new environments, languages, and automation tools and processes that need to be implemented.

6.2.1.1 Quantifying Elasticity

Reference [1] provides a great explanation of the core elasticity metrics and explains the difference between Elasticity, Scalability and Efficiency. There is considerable confusion in the literature, especially when it comes to the first two concepts.

Figure 6.4 (taken from [1]) helps understand the definition of the core elasticity metrics. These metrics, presented in Equations 6.1 through 6.6 have been included in benchmark definitions.

1. T_u is the accumulated time in underprovisioned state

$$T_u = \sum_{i}^{6} A_i \tag{6.1}$$

2. $\bar{R}_u$ is the average amount of underprovisioned resources during an underprovisioned period

$$\bar{R}_u = \frac{1}{6} \sum_{i}^{6} U_i \tag{6.2}$$

3. $R_u$is the accumulated amount of underprovisioned resources

$$R_u = \sum_{i}^{6} U_i \tag{6.3}$$

4. T_o is the accumulated time in overprovisioned state

$$T_o = \sum_{i}^{6} B_i \tag{6.4}$$

5. $\bar{R}_o$ is the average amount of overprovisioned resources during an overprovisioned period

$$\bar{R}_o = \frac{1}{6} \sum_{i}^{6} O_i \tag{6.5}$$

6. $R_o$is the accumulated amount of overprovisioned resources

$$R_o = \sum_{i}^{6} O_i \tag{6.6}$$

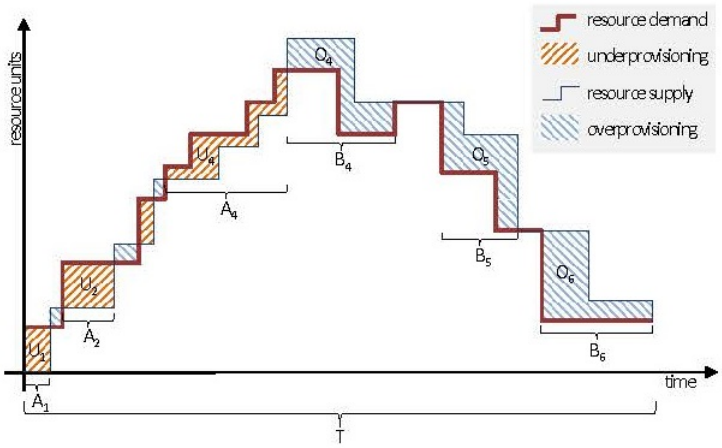


Fig. 6.4: Capturing Core Elasticity Metrics.

6.2.2 Availability

Availability is an “easy” technical requirement - it always shows in every provider’s SLAs and has a crisp mathematical definition that all providers use. Equation 6.7 defines the availability metric A of an offered service as

$$A = \frac{T_a - T_d}{T_a} \text{x} 100\%, \tag{6.7}$$

where T_a is an *agreed service time period* and T_d is the *downtime*, i.e., the amount of time during T_a that the service is not available.

In typical SLAs, providers promise "Five 9s", that is 99.999% availability. Figure 6.5 (taken from [4]) gives you an idea about the actual amount of time percentages translate into.

bmc

The Nines of Availability

Availability percentages vs service downtime

Availability %	Downtime per year	Downtime per month	Downtime per week
90% (one nine)	36.5 days	72 hours	16.8 hours
99% (two nines)	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.9% (three nines)	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (four nines)	52.56 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% (seven nines)	3.15 seconds	0.259 seconds	0.0605 seconds

Fig. 6.5: Availability in percent and absolute numbers.

In typical SLAs, providers set T_a to a month. This is not necessarily ideal for consumers, since it averages out peak periods with quiet ones. The impact of the downtime occurring during a peak period cannot simply be offset with the service being available during a period with no traffic hitting the service. In the industry, this is described semi-artistically with the "watermelon effect" graphic, shown in Figure 6.6. The SLA availability numbers are great, but the consumer is not happy.

One last observation about SLAs that mention this technical availability: all providers offer financial credits in case they fail to meet their obligation. The credit appears in the next billing cycle - this conveniently (for the provider) assumes that the consumer is still staying put.

6.2.3 Observability

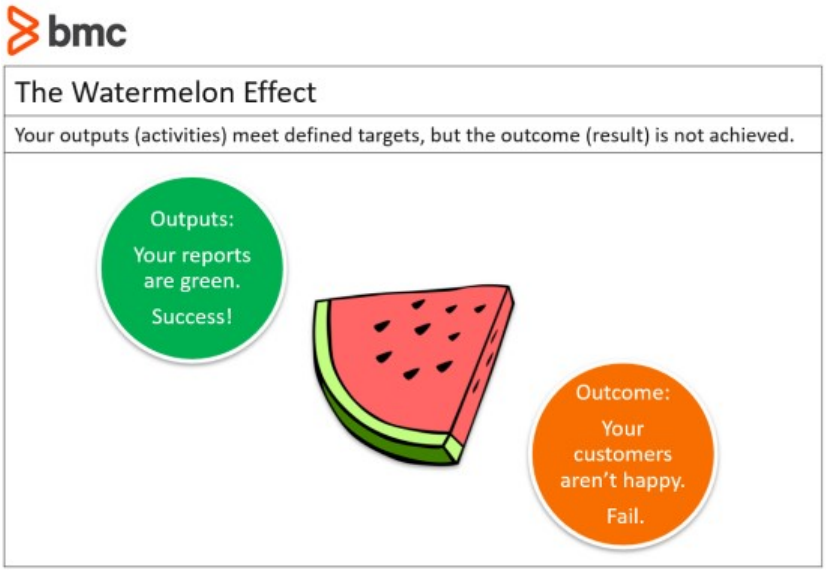


Fig. 6.6: "The operation was successful but the patient has died..." (taken from [4]).

—— This section is a stub for this semester. ——

—— We will cover this topic in detail in the spring 2024 semester. ——

6.2.4 Manageability

This technical requirement is quite abstract and difficult to quantify. It expresses the administrators' desire to avoid nightmares in operating and troubleshooting. Note that administrators in both provider as well as consumer enterprises prefer architectural solutions that are easy to manage.

What knobs does the architect have at her/his disposal to keep the administrator compadres happy? Automation is the most powerful one.

6.2.5 Performance

Performance is a "tough" technical requirement - it never shows in provider's SLAs.

There are three main reasons for that. First, as we have alluded to in Section 4.1.1.4, performance can be defined in several ways, based on the type of workload the service is handling. Second, too many factors affect even the "simplest"

metrics of performance. There are too many moving parts to orchestrate. And last but not least, there are too many cooks in the kitchen.

Example 6.4. **Delay as a performance metric.** Consider a SaaS service. Suppose that users of the service desire delay as their performance metric. Roughly speaking, from the user's perspective, a definition of delay D that makes sense would be given by

$$D = T_r - T_s$$

where T_r is the time instant a response from the service came to the user and T_s is the time instant at which the user has sent their request to the service.

Regarding the first reason, observe that D is a random variable. So, a KPI that uses delay can use:

1. D itself. This has appeared in SLAs that provide absolute delay guarantees.
2. The average value of D
3. The jitter of D.
4. Percentiles of D. This has appeared in Google SRE environments.

Regarding the second and third reasons, observe that in the cloud computing environment, D consists of at least three components.

1. $D_{consumer}$: this is the delay portion incurred while the request as well as its response spend inside the consumer's environment.
2. $D_{carrier}$: this is delay inside the carrier's environment (remember this actor from our discussions in Section 1.2.1.5, page 9?)
3. $D_{provider}$: this is delay inside the provider's environment

So, there are three types[5] of cooks in the kitchen. Let's take a look at the provider's kitchen, and see what factors affect this delay. We know the request and response will traverse the switch fabric at least twice; if the SaaS code is implemented as microservices, we know that delay in several containers will occur.

Hmmmmmm... No wonder delay does not have a space in SLAs. △

6.2.6 Security

There are many, many technical requirements bundled inside a consumer's "Operations should be secure" wish. To name just a few:

1. Authentication
2. Data security
3. Compliance to policies

Each one requires different knobs. We will not deal much with security in this course; we have plenty of specialized security courses at NC State.

[5] In a multicloud environment, there are potentially more than four cooks altogether.

6.3 Tradeoffs

There is a *negative connotation* associated with tradeoffs in general. This is unfortunate: an architect or engineer who is a master of tradeoffs should be highly regarded! If one believes that there is no free lunch in engineering, one should love (and learn how to make and evaluate) tradeoffs.

6.3.1 What necessitates tradeoffs in a design?

Let's see first why there is no free lunch in an architect's life.

6.3.1.1 Culprit #1: Multiple objectives

Example 6.5. **Just two objectives can mess things up.** Consider two simple, separate minimization objectives, as shown in the equations below.

$$\min x^2, \quad \min(x-1)^2, \;\; x \in (-\infty, \infty)$$

It is **impossible to minimize both objectives**, since the first function achieves its minimum when $x = 0$ while the second one does so when $x = 1$. Choosing *any value* for x represents a tradeoff. △

The example is trivial, but the lesson it delivers for the (cloud or any other) architect is not. What's the lesson? We'll say it two ways:

(First way.) When designing for multiple objectives, we cannot have our pie and eat it too...

(Second way.) Damned if you do and damned if you don't...

Tradeoffs in the case of multiple objectives "sneak in" when one evaluates a design that was targeted at one (set of) objective(s) after the fact, for a different objective[6].

Example 6.6. **BGP and delay performance.** BGP, the protocol that keeps the forwarding tables in the Internet upto date, is criticized for finding paths through the internet that are "not the best in terms of delay".

This should come to no surprise to anyone who read the objective of this protocol. We quote from section 2 in RFC 1771, for convenience:

[6] This happens quite frequently in practice and is abused by marketing teams of competitor organizations.

> The primary function of a BGP speaking system is to exchange network reachability information with other BGP systems. This network reachability information includes information on the list of Autonomous Systems (ASs) that reachability information traverses. This information is sufficient to construct a graph of AS connectivity from which routing loops may be pruned and some policy decisions at the AS level may be enforced.

Nowhere it is mentioned that BGP considered delay as a primary, secondary or even tertiary function. △

6.3.1.2 Culprit #2: Constraints of all kinds

Example 6.7. **Constraints can mess it up too.** Designs (as well as implementations) very rarely happen in a vacuum. Typically you do not build up something from scratch. You don't have unlimited amounts of time to come up with the best solution, when a competitor is out in the market already or the CEO wants it yesterday (but can wait till tomorrow). You don't have unlimited pools of $$ either.

Even when designing for a single objective, a tradeoff may be unavoidable. △

6.3.2 How to evaluate tradeoffs?

So, as architects, we got to live with tradeoffs. The silly question, then, is given in the title of this section.

Unfortunately, to the best of the authors' knowledge, there is no systematic way to evaluate tradeoffs in a cloud computing environment. We have not seen best practices published either.

We recommend reading reference [5] - and see if you can apply it on any of the specific designs you come up with. We quote (slightly paraphrasing) from [5]:

> How do you make trade-offs when comparing such widely disparate things? In the past, decision makers have relied mostly on instinct, common sense, and guesswork. They've lacked a clear, rational, and easy-to-use trade-off methodology. To help fill that gap, we have developed a system—which we call **even swaps** —that provides a practical way of making trade-offs among any set of objectives across a range of alternatives. In essence, the even-swap method is a form of bartering—**it forces you to think about the value of one objective in terms of another**.
>
> The even-swap method will not make complex decisions easy; you'll still have to make hard choices about the values you set and the trades you make. What it does provide is a reliable mechanism for making trades and a coherent framework in which to make them. By simplifying and codifying the mechanical elements of trade-offs, the even-swap method lets you focus all your mental energy on the most important work of decision making: deciding the real value to you and your organization of different courses of action.
>
> Before you can begin making trade-offs, you need to have a clear picture of all your alternatives and their consequences for each of your objectives. A good way to create that picture is to draw up a **consequences table**. (This is the heart of the methodology.)
>
> a consequences table puts a lot of information into a concise and orderly format that allows you to compare your alternatives easily, objective by objective. It gives you a clear frame-

work for making trade-offs. Moreover, it imposes an important discipline, forcing you to define all alternatives, all objectives, and all relevant consequences at the outset of the decision process.

6.4 Migrating applications to the cloud

This is a problem for the solutions architect role we described in Section 6.1.2. Recall that this role works for the consumer actor.

6.4.1 Problem description and objectives

In a nutshell, as the name implies, the problem is conversion of applications from an initial environment (e.g., on premise consumer servers) to some form of service in the cloud. Note that the application to be migrated may or may not be in cloud native form. Any one of the generic technical requirements can be associated with this problem.

Migrating applications should not be confused with the related but more challenging problem of Cloud Migration. This is the transfer of the entire set of IT resources of a consumer from private servers and in-house data center facilities to public cloud architecture.

We quote from [23]:

What is application migration? Application migration is the process of moving software applications from one computing environment to another. This can include migrating applications from one data center to another, such as from a public to a private cloud, or from a company's on-premises server to a cloud provider's environment.

Organizations migrate applications to the cloud to take advantage of an improved cost structure, responsive scalability, and the ability to quickly update apps to meet changing demands.

3 steps to simplify application migration to the cloud The migration process can be complex, especially for businesses without experience moving applications to the cloud. However, with proper planning and execution, it is possible to achieve a smooth migration. Here are three steps that can help:

1. **Assemble the right team.** Recruit key representatives from each business unit involved in the migration project. These individuals will help to inform and support the project as it moves forward. It's also important to put together a team of internal technical experts to assist with the migration at each phase, even if your organization is working with a third-party partner.

2. **Win over your stakeholders.** Present the business case for the migration—whether it be cost savings, streamlining operations, a competitive market edge, or all of the above. In getting approval from leadership, you'll want to agree on the project goals, budget, and timeline. Be sure to keep your stakeholders informed as the project moves forward and through each phase.

3. **Audit your applications.** As you audit your application landscape, evaluate and group applications according to the following criteria:

• Is this a business app or technical app?
• Is the app modern or legacy?
• Will refactoring or rewriting be required?
• What is the downtime sensitivity around this app?
• Was the app developed in-house or by a third party?

When assessing each application, it is also critical to identify dependencies, integrations, and technical requirements. You'll want to understand the application's architecture, security policies, and the tools and software used to manage access, performance, and troubleshooting. Prioritize migrations based on these criteria, as well as on the business needs and your budget. Then decide which cloud to migrate to on an application-by-application basis. With your applications grouped logically and an understanding of where each app is going, you can complete your application migration plan.

Application migration best practices Here are a few best practices for planning and implementing an application migration:

• Keep the business goals and end-state objectives clear to everyone involved. Establishing and reaffirming the purpose, benefits, and end results of the migration will help everyone stay motivated and focused.

• Start small and minimize risk. Starting with just one application (or a small group of non-critical applications) allows team members to gain confidence, identify potential issues, and show results. Another way to minimize risk is to practice migrations on test apps within test environments.

• Bring in outside experts and third-party tools to supplement in-house capabilities. Migrating apps is a complicated process, and when not executed properly, it has the potential for costly errors and data loss. It's smart to make the investment in additional tools and expertise to make sure you're adequately prepared.

Common application migration strategies There are a variety of strategies for effectively migrating applications to the cloud. The right choice will depend on business needs, budget constraints, and other factors. Here are some of the most common options:

• Move without conversions. Moving a primary asset, such as a website, from an on-premises environment to the public cloud—without changing anything about the asset—can result in significant savings. Using a public cloud also allows for additional support through built-in cloud optimization functionality like disaster recovery and on-demand capacity extension.

• Choose a SaaS replacement. Finding an existing marketplace Software-as-a-Service (SaaS) offering that can already do what you need can reduce the burden of the migration process as well as free up on-premises resources for other workloads.

• Choose a PaaS replacement. Adopting an existing Platform-as-a-Service (PaaS) without adjusting any architecture can replace expensive on-premises server needs with a subscription-based service. A PaaS replacement often leads to increased agility during periods of high demand.

• Re-architect. Modernizing an application or service through cloud migration can add new and improved functionality, add tangible product value, and give new life to an older but still valuable product.

• Retire. If a given workload doesn't provide any business value, and it isn't intrinsic to another workload, retire it.

What is the application migration process? A standard application migration includes the following steps:

1. **Plan:** Review and assess your applications, business goals, and teams to create a plan for the migration. Also consider additional tools. There are a variety of options for third-party application migration software and services. These tools can help with managing and moving data between platforms, as well as providing in-depth data analysis and monitoring.

2. **Test:** Before performing any actual migrations, use a mock migration to perfect the process. And then, after each real migration phase, test whatever has been migrated into

the new environment and document the outcomes. Regular testing and sandboxing allow the team to catch problems early and regroup or change direction before data is lost and progress is wasted.

3. **Migrate in waves:** It's best to group applications and then perform the migration in phases. Document each phase using a project management tool to keep everyone, including stakeholders, informed and to gather supporting documentation.

4. **Follow up:** Once the migration is complete, perform follow-up tests to determine whether cloud migration was correctly executed. This includes analyzing application performance, looking for potential disruptions, and reviewing database security.

What does a successful application migration to cloud look like? An enterprise that successfully migrates its applications to the cloud can achieve the following benefits:

- Improved and modernized solutions for business goals
- Reduced time allocated for training new employees
- Wider access to distributed applications
- Decreased complexity and costs
- Application consolidation
- Better security and management
- Increased productivity
- Extended value of IT investments

Example 6.8. **Migrating legacy, monolithic applications.** (Example adapted from [30]). Consider a typical ecommerce application. Its logic might contain a web server, a load balancer, a catalog service that services up product images, an ordering system, a payment function, and a shipping component. The implementation is monolithic if the the logic is implemented as a single code base.

Migrating this application to the cloud may be done "as is". The same code may be accessed now as a SaaS. It may, however, be split into microservices, each one running on its own container. The code can still be accessed as SaaS. △

6.4.2 Potential solutions and tradeoffs

Reference [23] does not describe a "technical solution" in any detail. The documents in `https://cloud.google.com/architecture/migration` do exactly that; they describe the process suggested by Google for migrating workloads as well as datasets to GCP.

We quote from `https://cloud.google.com/architecture/migration`:

This document is useful for the following migration scenarios:
From an on premises environment
From a private hosting environment
From another cloud provider to Google Cloud

As part of your migration journey, you have to make decisions that are dependent on the environment, the workloads, and the infrastructure that you're migrating to Google Cloud or to a hybrid cloud environment. These documents help you choose the best path to suit your migration needs in the following ways:
Establish a framework to design and run your migration journey by using the Migrate to Google Cloud series.
Use this framework as a baseline against which you can assess your migration progress.

Give guidance that's specific to a particular environment or use case by building on the Migrate to Google Cloud framework, such as Migrate VMs with Migrate to VMs, Migrate containers to Google Cloud, and Migrate VMs to containers with Migrate to Containers.

Benefits of establishing a migration framework Establishing a migration framework is important because migration can be a repeatable task. For example, if you initially migrate your VMs to Google Cloud, you might also consider moving other data and workloads to Google Cloud. Establishing a general framework that can be applied to different workloads can make future migrations easier for you.

The diagram in Figure 6.7 illustrates the migration phases:

Fig. 6.7: The migration path phases, according to GCP.

During each migration step, you follow the phases defined in `https://cloud.google.com/architecture/migration-to-gcp-getting-started`:
Assess and discover your workloads.
Plan and build a foundation.
Deploy your workloads.
Optimize your environment and workloads.

This journey isn't unique to Google Cloud. Moving from one environment to another is a challenging task, so you need to plan and execute your migration carefully. No matter what you're migrating—whether apps, VMs, or containers—you need to complete tasks such as creating an inventory, establishing user and service identities, deploying your workloads, and optimizing for performance and scalability.

We ask you to investigate the steps involved in the first phase in Problem 6.7. We ask you to evaluate specific metrics associated with data migration in Problem 6.8.

6.5 Tenant collocation

This is a major problem for the provider (master) architect we described in Section 6.1.1. The provider sells cloud services to a (large) number of consumers. For the purposes of this problem, we will call a consumer a tenant. There are several problems associated with tenants coexisting in a shared environment, be it the premises of a cloud provider - or an apartment landlord.

It is also a problem for the consumer architect, in case the consumer also supports tenants. Consider, for example, an enterprise (like Snowflake, `www.snowflake.com`) that gets IaaS from, say, AWS and then provides database PaaS/SaaS services to its own customers (i.e., tenants).

6.5.1 Problem description and objectives

In a nutshell, the problem for the cloud provider can be described as follows:

Provide each tenant their own, private cloud environment.

The following requirements relate to this problem:

1. **Tenant identification**. Tenants must be identified uniquely, if nothing else, for billing purposes. Requests (that is, network traffic) entering the provider's datacenter must be identified as belonging to tenant X.
2. **Tenant isolation**. This term refers to the need to separate tenant traffic in servers, switch fabric, storage, and interconnect networks. It would be embarrassing, let alone a legal liability to mix traffic from two tenants.
3. **Sharing cloud resources among tenants**. While tenants request and pay for use of compute and storage resources, they cannot request switch fabric resources.
4. **Protecting provider resources from misbehaving tenants**. The financial agreement between a consumer and the provider specifies terms of use. What happens if a tenant willingly or not sends traffic to the datacenter in violation of these terms?

Note that the first two requirements are necessary and sufficient for solving the stated problem. The remaining ones describe additional objectives.

6.5.2 Potential solutions and tradeoffs

We limit our discussion to the tenant identification question.

6.5.2.1 Tenant identification

The following two examples provider answers to the tenant identification question.

Example 6.9. **NVO3 virtualization of switch fabrics.** When the switch fabric is virtualized according to NVO3 guidelines, the tenant traffic contains a unique identifier. When NVO3 is implemented with the VxLAN protocol, the identifier is the 24-bit *VNID*. △

Example 6.10. **Server virtualization.** Inside a server, the tenant traffic also needs a unique identifier. This one may be different from the VxLAN VNID. For example, in Kubernetes, the tenant is identified with a unique *namespace*. (More on that in the next chapter.) △

Choosing VxLAN VNID and Kubernetes namespace as the tenant identifiers requires a mapping between the two. This is a tradeoff that needs to be made.

6.5.2.2 Tenant isolation

Tenant isolation inside a compute server is provided via the separate allocation of containers or VMs; isolation inside a storage server is provided by allocation of separate storage "blocks". Isolation on the switch fabric is provided by VxLAN.

6.5.2.3 Sharing cloud resources among tenants

Sharing of compute and storage resources is provided by the OS. Sharing of the switch fabric (bandwidth) resource is provided by the LSA of the routers.

6.5.2.4 Protecting provider resources from misbehaving tenants

Protection of a compute and storage resource from misbehaving tenants is also provided by the OS. Protection of the switch fabric (bandwidth) resource is provided by the LSA of the routers.

6.5.2.5 An alternative design

We only discuss an alternative for tenant identification. Instead of the VNID used in VxLAN, one can choose a a 16-bit unsigned integer as a starting point. Then map this integer to whatever identifiers the switch fabric, storage interconnect and server virtualization technologies use.

6.6 Virtual Private Cloud Networking (VPCN)

This is a problem for the network architect role we described in Section 6.1.3. This is the design of only the sixth type of network we mentioned in that section, so the architect works for the consumer actor.

6.6.1 Problem description and (a minimal set of) objectives

In a nutshell, the problem can be loosely described as follows:

We are given a number of containers and/or VMs that can vary in time, since autoscaling can occur. We want to "connect all of them together", in a layer 3 network.

The following (incomplete list of) requirements specify in more concrete, technical terms what connectivity may mean:

(R1) the VPCN must be completely isolated from networks of other consumers (tenants);
(R2) nodes in the VPCN should be able to access any provider service;
(R3) nodes in the VPCN should be able to access the consumer's external sites;
(R4) the containers must reside in their own IPv4 subnet;
(R5) the VMs must reside in their own IPv4 subnet;
(R6) load on the containers should be balanced.

6.6.2 Potential solutions and tradeoffs

6.6.2.1 A potential solution

Here are the basic building blocks of a solution:

(B1) Use a gateway router for connectivity to the consumer's external sites.
(B2) Use a router for connectivity to provider services.
(B3) Use IPv4 subnet 10.0.0.0/16 allowing for 65K containers.
(B4) Use IPv4 subnet 10.1.0.0/24 allowing for 256 VMs.
(B5) Use a load balancer service provided by the provider.

How does the architect assure her/his boss that the requirements are met? Satisfaction of:

(R1) is assured by the provider.
(R2) is assured by the provider and (B2).
(R3) is met by (B1).
(R4) is met by (B3).
(R5) is met by (B4).
(R6) is assured by the provider or the tenant.

The solutions offered by Amazon AWS and Google GCP are described in great detail in references [6] and [8] respectively. Reference [7] provides a summary of the AWS VPC solution.

6.6.2.2 Description of tradeoffs

The obvious tradeoffs of our proposed solution come from the selected size of the two subnets. Another one comes from the choice of a single gateway router. This design does not provide availability; that should have been expected, however, given that the set of stated requirements did not include availability. A last tradeoff comes from the choice of using provider-supplied load balancers.

6.6.2.3 An alternative design

A number of such designs can be produced, based on what tradeoffs we want to change. For example, use of two gateway routers instead of one; using a custom-made load balancer.

6.7 Load distribution and balancing

This is a problem for the solutions architect role we described in Section 6.1.2. Recall that this role works for both the provider as well as the consumer actor.

6.7.1 Problem description and objectives

For some fundamental notions, definition of load, common metrics for load and the difference between load distribution and load balancing, see chapters 1 and 2 in the ECE/CSC577 notes on [2].

6.7.2 Potential solutions and tradeoffs

For a general description of load balancing types, see [3]. For specific solutions employed in Google's GCP, see [9]; for an in depth look at Maglev, a network load balancer in the google solution set, see [10]; the site in [11] is a one-stop shop for Google load balancing solutions, with a plethora of documentation, tutorials, use cases and sample code. For specific solutions employed in AWS, see [12].

For a brief description of DNS-based load balancing, see [13]; for limitations of this form of load balancing, see the section "Limitations of DNS-based load balancing" in [14].

Facebook's Katran load balancer is described in [15].

6.8 Cloud (Resource) Orchestration

—— This section is a stub for this semester. ——

This is a problem for the solutions architect role we described in Section 6.1.2. Recall that this role works for both the provider as well as the consumer actor.

6.8.1 Problem description and objectives

We quote from [16]:

What is Cloud Orchestration? Cloud Orchestration is the process of automating the tasks needed to manage connections and operations of workloads on private and public clouds. Cloud orchestration technologies integrate automated tasks and processes into a workflow to perform specific business functions.

Cloud orchestration tools entail policy enforcement and ensure that processes have the proper permission to execute or connect to a workload. Typical cloud orchestration tasks are to provision or start server workloads, provision storage capacity as needed, and instantiate virtual machines (VMs) by orchestrating services, workloads, and resources in the cloud.

Cloud automation and orchestration tools help to reduce the challenges organization have had deploying automation tools by eliminating islands of automation in favor of a cohesive, cloud-wide approach that encompasses both public cloud and private cloud components.

Why is Cloud Orchestration important? The rapid adoption of containerized, microservices based applications that communicate via APIs has created the demand for automation of deploying and managing applications across the cloud. This increasing complexity has created the demand for cloud orchestration software that can manage the myriad dependencies across multiple clouds, with policy-driven security and management capabilities.

As organizations increasingly adopt a hybrid cloud architecture, the need for both public cloud orchestration and hybrid cloud orchestration has continued to grow.

Most importantly, cloud orchestration reduces the need for IT staff to manually handle automation tasks, freeing up resources for more productive works. This also reduces the opportunity for manual errors to occur. This lets organizations spend time on innovation, enabling accelerated deployment of applications across hybrid IT infrastructure, orchestrating various processes across domains and systems. The result is an improved experience for users and customers enterprise-wide.

What are Benefits of Cloud Orchestration? Cloud orchestration simplifies automation across a hybrid cloud environment while ensuring that policies and security protocols are maintained in a dynamic, modern IT environment. Cloud orchestration reduces overall costs while accelerating delivery of services, automates management and coordination of complex hybrid environments, eliminates provisioning errors, and enables self-service provisioning of services without the need for IT intervention.

Cloud orchestration can also help prevent VM sprawl by enhancing the visibility into resource usage across clouds. Other high-level benefits include automating connections between workloads to ensure links are configured properly, and many cloud orchestration platforms do so by the use of a web-based portal that offers single pane of glass management organization-wide.In advanced organizations, developers and line-of-business workers can turn to cloud orchestration software as a self-service mechanism to deploy resources; administrators can use it to track the organization's reliance on various IT offerings and manage chargebacks.

Detailed benefits of cloud orchestration include:

• Improving efficiency of resource utilization, eliminate over-provisioning.

• Monitor, alert, and report on unexpected conditions to diagnose root cause.

• Simplify data integrations and automatically apply policies for governance and security.

• Manage dependencies across clouds to ensure proper execution of tasks.

• Integrate existing identity and access management systems as part of organization-wide security to ensure only authorized users and applications can access or modify automations.

• Eliminate need to build ad-hoc tools when new automations are required.

- Deliver workflow tools for managing and scheduling for IT and line of business users.
- Provide the bridge between clouds or between private and public environments.

6.8.2 Potential solutions and tradeoffs

We quote again from [16]:

> **What are Cloud Orchestration Models?** Cloud Director orchestration models let you provision a ready environment to deploy virtual servers. Orchestration can be either single cloud or multi-cloud. In a single cloud model, multiple applications run on the same cloud service provider, which is a simpler setup. The more complicated, but also more powerful model is the multi-cloud setup. Here we have multiple applications, which are located on different cloud platforms, and multi-cloud orchestration interconnects them so they can perform as a single system, with the advantage of high redundancy
>
> There are also three delivery models for cloud services in general, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS)
>
> IaaS is most commonly utilized in cloud orchestration environments. IaaS providers offer network hardware, storage, and servers, as well as physical security in either a dedicated or multi-tenant environment. IaaS providers also offer virtualization services and orchestration tools that can streamline IT operations within their cloud or between multiple clouds.
>
> PaaS providers also provide operating systems and middleware, and SaaS providers deliver applications only, through a web interface, typically on a subscription basis.
>
> Cloud orchestration tools enable all these models to operate as one, providing automation across models and across clouds, but typically taking advantage of IaaS providers to automate the deployment process, reducing labor and resources required so they may focus on bottom-line generating functions.
>
> **What is Cloud Orchestration vs Cloud Automation?** Orchestration is a superset of automation. Cloud orchestration goes beyond automation to also provide coordination between multiple automated tasks. Automation, on the other hand, has the goal of enabling one task, such as initiating a web server, to be repeated or iterated quickly with little manual intervention. Thus, cloud orchestration focuses on the entirely of the IT processes, automation on a single piece.
>
> Practically, cloud orchestration and automation work hand in hand to ensure that cloud-based applications and services are deployed and delivered in a cohesive, functional, cloud environment that operates efficiently and cost effectively.
>
> Cloud orchestration adds enterprise-wide functions such as redundancy, scalability, failover and fail-back, manages dependencies, and bundles them into a single package, greatly reducing the overall IT burden. Orchestration additionally offers enhanced visibility into the resources and processes in use, which can help prevent VM sprawl and help organizations track resource usage by department, business unit, or even by individual user. Orchestration enables automatic scaling of distributed applications, disaster recovery and business continuity, and inter-service configuration.
>
> Cloud orchestration is increasingly important due to the growth of containerization, which facilitates scaling applications across clouds, both public and private. Tools such as VMware Tanzu help facilitate container orchestration across an organization's entire cloud presence.

6.8.2.1 Service mesh solutions and tradeoffs

—— This section is a stub for this semester. ——

—— We will cover this topic in detail in the spring 2024 semester. ——

6.9 Data ingestion

—— This section is a stub for this semester. ——

—— We will cover this topic in detail in the spring 2024 semester. ——

Most organizations have more data on hand than they know what to do with—but collecting this information is only the first step. To make the most of your enterprise data, you need to migrate it from one or more sources, and then transfer it to a centralized data warehouse for efficient analysis and reporting.

The term "data ingestion" refers to any process that transports data from one location to another so that it can be taken up for further processing or analysis. In particular, the use of the word "ingestion" suggests that some or all of the data is located outside your internal systems. The two main types of data ingestion are:

- Batch data ingestion, in which data is collected and transferred in batches at regular intervals.
- Streaming data ingestion, in which data is collected in real-time (or nearly) and loaded into the target location almost immediately.

Both batch and streaming data ingestion have their pros and cons. Streaming data ingestion is best when users need up-to-the-minute data and insights, while batch data ingestion is more efficient and practical when time isn't of the essence.

Data ingestion is similar to, but distinct from, the concept of data integration, which seeks to integrate multiple data sources into a cohesive whole. With data integration, the sources may be entirely within your own systems; on the other hand, data ingestion suggests that at least part of the data is pulled from another location (e.g. a website, SaaS application, or external database).

6.9.1 ETL

The term ETL (extract, transform, load) refers to a specific type of data ingestion or data integration that follows a defined three-step process:

First, the data is extracted from a source or sources (e.g. files, databases, SaaS applications, or websites). Next, the data is transformed according to specific business rules, cleaning up the information and structuring it in a way that matches the schema of the target location. Finally, the data is loaded into the target location. This may be a data warehouse (a structured repository for use with business intelligence and analytics) or a data lake (a very large repository that can accommodate unstructured and raw data). ETL is one type of data ingestion, but it's not the only type.

ELT (extract, load, transform) refers to a separate form of data ingestion in which data is first loaded into the target location before (possibly) being transformed. This alternate approach is often better suited for unstructured data and data lakes, where not all data may need to be (or can be) transformed.

6.9.2 Problem description and objectives

6.9.2.1 Distinguish betwen ETL and Data Ingestion

A problem is that many people confuse ETL with Data Ingestion. ETL is a special case of data ingestion that inserts a series of transformations in between the data being extracted from the source and loaded into the target location.

Just a few different types of ETL transformations are:

- Aggregation: Merging two or more database tables together.
- Cleansing: Removing information that is inaccurate, irrelevant, or incomplete.
- Deduplication: Deleting duplicate copies of information.
- Joining: Combining two or more database tables that share a matching column.
- Splitting: Dividing a single database table into two or more tables.
- Summarization: Creating new data by performing various calculations (e.g. summing up the revenue from each sales representative on a team).
- Validation: Ensuring that the data is accurate, high-quality, and using a standard format (e.g. converting all timestamps into Greenwich Mean Time).

Data ingestion acts as a backbone for ETL by efficiently handling large volumes of big data, but without transformations, it is often not sufficient in itself to meet the needs of a modern enterprise. Organizations cannot sustainably cleanse, merge, and validate data without establishing an automated ETL pipeline that transforms the data as necessary.

Some newer data warehouse solutions allow users to perform transformations on data when it's already ingested and loaded into the data warehouse. So why then is ETL still necessary?

In fact, ETL, rather than data ingestion, remains the right choice for many use cases. For example, ETL is likely preferable to raw data ingestion if you'll be querying the data over and over, in which case you'll only need to transform the data once before loading it into the data warehouse. In-warehouse transformations, on the other hand, need to transform the data repeatedly for every ad hoc query that you run, which could significantly slow down your analytics runtimes.

6.9.2.2 Data Ingestion vs ETL use cases

ETL has a wide variety of possible data-driven use cases in the modern enterprise. According to a study by McKinsey & Company, for example, businesses that in-

tensively use customer analytics are 23 times more likely to succeed at customer acquisition, and 19 times more likely to be highly profitable.

One popular ETL use case: sales and marketing departments that need to find valuable insights about how to recruit and retain more customers. Because these teams have access to a great deal of data sources, from sales calls to social media, ETL is needed to filter and process this data before any analytics workloads can be run.

ETL is also widely used to migrate data from legacy systems to new IT infrastructure. ETL solutions can extract the data from a source legacy system, transform it as necessary to fit the new architecture, and then finally load it into the new system.

The transformation stage of ETL is especially important when combining data from multiple sources. Transformations such as data cleansing, deduplication, summarization, and validation ensure that your enterprise data is always as accurate and up-to-date as possible.

Data Ingestion Use Cases Because big data is characterized by tremendous volume, velocity, and variety, the use cases of data ingestion (without transformation) are rarer. For example, data ingestion may be used for logging and monitoring, where the business needs to store raw text files containing information about your IT environment, without necessarily having to transform the data itself.

With a bit of adjustment, data ingestion can also be used for data replication purposes as well. Data replication is the act of storing the same information in multiple locations (e.g. different servers or nodes) in order to support the high availability of your data. In the event that one of the servers or nodes goes down, you can continue to access the replicated data in a different location.

There's only a slight difference between data replication and data ingestion: data ingestion collects data from one or more sources (including possibly external sources), while data replication copies data from one location to another. Because data replication copies the data without transforming it, ETL is unnecessary here and we can simply use data ingestion instead.

6.9.3 Potential solutions and tradeoffs

6.10 Cloud Infosec

This is a problem for the security architect role we described in Section 6.1.7. Recall that this role works for both the provider as well as the consumer actor.

Cloud security is a responsibility that is shared between the cloud provider and the customer. There are basically three categories of responsibilities:

- responsibilities that are always the provider's
- responsibilities that are always the customer's

- responsibilities that vary depending on the service model: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS), such as cloud email.

The security responsibilities that are always the provider's are related to the safeguarding of the infrastructure itself, as well as access to, patching, and configuration of the physical hosts and the physical network on which the compute instances run and the storage and other resources reside.

The security responsibilities that are always the customer's include managing users and their access privileges (identity and access management), the safeguarding of cloud accounts from unauthorized access, the encryption and protection of cloud-based data assets, and managing its security posture (compliance).

6.10.1 Cloud Security Challenges

6.10.1.1 Increased Attack Area

The public cloud environment has become a large and highly attractive attack surface for hackers who exploit poorly secured cloud ingress ports in order to access and disrupt workloads and data in the cloud. Malware, Zero-Day, Account Takeover and many other malicious threats have become a day-to-day reality.

6.10.1.2 Lack of Visibility

In the IaaS model, the cloud providers have full control over the infrastructure layer and do not expose it to their customers. The lack of visibility and control is further extended in the PaaS and SaaS cloud models. Cloud customers often cannot effectively identify and quantify their cloud assets or visualize their cloud environments.

6.10.1.3 Changing Workloads

Cloud assets are provisioned and decommissioned dynamically—at scale and at velocity. Traditional security tools are simply incapable of enforcing protection policies in such a flexible and dynamic environment with its ever-changing and ephemeral workloads.

6.10.1.4 DevSecOps & Automation

DevSecOps stands for development, security, and operations. It's an approach to culture, automation, and platform design that integrates security as a shared responsibility throughout the entire IT lifecycle.

Organizations that have embraced the highly automated DevOps CI/CD culture must ensure that appropriate security controls are identified and embedded in code and templates early in the development cycle. Security-related changes implemented after a workload has been deployed in production can undermine the organization's security posture as well as lengthen time to market.

6.10.1.5 Privileges and Key Management

Often cloud user roles are configured very loosely, granting extensive privileges beyond what is intended or required. One common example is giving database delete or write permissions to untrained users or users who have no business need to delete or add database assets. At the application level, improperly configured keys and privileges expose sessions to security risks.

6.10.1.6 Environments

Managing security in a consistent way in the hybrid and multicloud environments favored by enterprises these days requires methods and tools that work seamlessly across public cloud providers, private cloud providers, and on-premise deployments—including branch office edge protection for geographically distributed organizations.

6.10.1.7 Compliance and Governance

All the leading cloud providers have aligned themselves with most of the well-known accreditation programs such as PCI 3.2, NIST 800-53, HIPAA and GDPR. However, customers are responsible for ensuring that their workload and data processes are compliant. Given the poor visibility as well as the dynamics of the cloud environment, the compliance audit process becomes close to mission impossible unless tools are used to achieve continuous compliance checks and issue real-time alerts about misconfigurations.

6.10.2 Cloud Security on the Public Cloud

The architect has a lot of tools at their disposal from the major Cloud Service Providers to enable. Amazon Web Services (AWS), Microsoft Azure (Azure), and Google Cloud Platform (GCP) offer many cloud native security features and services. Supplementary third-party solutions are available to achieve enterprise-grade cloud workload protection from breaches, data leaks, and targeted attacks in the cloud environment. The following are some of the principles the architect can take:

6.10.2.1 Granular, policy-based IAM and authentication controls

Identity and Access Management (IAM) roles are entities you create and assign specific permissions to that allow trusted identities such as workforce identities and applications to perform actions in AWS. When your trusted identities assume IAM roles, they are granted only the permissions scoped by those IAM roles.

Work with groups and roles rather than at the individual IAM level to make it easier to update IAM definitions as business requirements change. Grant only the minimal access privileges to assets and APIs that are essential for a group or role to carry out its tasks. The more extensive privileges, the higher the levels of authentication. And don't neglect good IAM hygiene, enforcing strong password policies, permission time-outs, and so on.

6.10.2.2 Zero-trust cloud network security controls across logically isolated networks and micro-segments

Deploy business-critical resources and apps in logically isolated sections of the provider's cloud network, such as Virtual Private Clouds (AWS and Google) or vNET (Azure). Use subnets to micro-segment workloads from each other, with granular security policies at subnet gateways. Use dedicated WAN links in hybrid architectures, and use static user-defined routing configurations to customize access to virtual devices, virtual networks and their gateways, and public IP addresses.

6.10.2.3 Enforcement of virtual server protection policies and processes such as change management and software updates

This action applies governance and compliance rules and templates when provisioning virtual servers, auditing for configuration deviations, and remediating automatically where possible.

6.10.3 Safeguarding Cloud Web Apps

This will granularly inspect and control traffic to and from web application servers, automatically updates security rules in response to traffic behavior changes, and is deployed closer to microservices that are running workloads.

6.10.3.1 Enhanced data protection

Enhanced data protection with encryption at all transport layers, secure file shares and communications, continuous compliance risk management, and maintaining

good data storage resource hygiene such as detecting misconfigured buckets and terminating orphan resources.

6.10.3.2 Thread Intelligence

A cloud security solutions can add context to the large and diverse streams of cloud-native logs by intelligently cross-referencing aggregated log data with internal data such as asset and configuration management systems, vulnerability scanners, etc. and external data such as public threat intelligence feeds, geolocation databases, etc. They also provide tools that help visualize and query the threat landscape and promote quicker incident response times. AI-based anomaly detection algorithms are applied to catch unknown threats, which then undergo forensics analysis to determine their risk profile. Real-time alerts on intrusions and policy violations shorten times to remediation, sometimes even triggering auto-remediation workflows.

6.11 Disaster Recovery

—— This section is a stub for this semester. ——

—— We will cover this topic in detail in the spring 2024 semester. ——

This is a problem for the solutions architect role we described in Section 6.1.2. Recall that this role works for both the provider as well as the consumer actor.

6.11.1 Problem description and objectives

To start with, what is considered a *disaster*? The definition is not "standard", as we'll see. We quote from AWS-disaster:

> A disaster is an unexpected problem resulting in a slowdown, interruption, or network outage in an IT system. Outages come in many forms, including the following examples:
> An earthquake or fire
> Technology failures
> System incompatibilities
> Simple human error
> Intentional unauthorized access by third parties

What is then meant by *Disaster Recovery*? We quote from [18]:

> **What is Disaster Recovery?**
> **Disaster recovery** is an organization's method of regaining access and functionality to its IT infrastructure after events like a natural disaster, cyber attack, or even business

disruptions related to the COVID-19 pandemic. A variety of disaster recovery (DR) methods can be part of a disaster recovery plan. DR is one aspect of business continuity.

How does disaster recovery work?

Disaster recovery relies upon the replication of data and computer processing in an off-premises location not affected by the disaster. When servers go down because of a natural disaster, equipment failure or cyber attack, a business needs to recover lost data from a second location where the data is backed up. Ideally, an organization can transfer its computer processing to that remote location as well in order to continue operations.

5 top elements of an effective disaster recovery plan

1. Disaster recovery team: This assigned group of specialists will be responsible for creating, implementing and managing the disaster recovery plan. This plan should define each team member's role and responsibilities. In the event of a disaster, the recovery team should know how to communicate with each other, employees, vendors, and customers.

2. Risk evaluation: Assess potential hazards that put your organization at risk. Depending on the type of event, strategize what measures and resources will be needed to resume business. For example, in the event of a cyber attack, what data protection measures will the recovery team have in place to respond?

3. Business-critical asset identification: A good disaster recovery plan includes documentation of which systems, applications, data, and other resources are most critical for business continuity, as well as the necessary steps to recover data.

4. Backups: Determine what needs backup (or to be relocated), who should perform backups, and how backups will be implemented. Include a recovery point objective (RPO) that states the frequency of backups and a recovery time objective (RTO) that defines the maximum amount of downtime allowable after a disaster. These metrics create limits to guide the choice of IT strategy, processes and procedures that make up an organization's disaster recovery plan. The amount of downtime an organization can handle and how frequently the organization backs up its data will inform the disaster recovery strategy.

5. Testing and optimization: The recovery team should continually test and update its strategy to address ever-evolving threats and business needs. By continually ensuring that a company is ready to face the worst-case scenarios in disaster situations, it can successfully navigate such challenges. In planning how to respond to a cyber attack, for example, it's important that organizations continually test and optimize their security and data protection strategies and have protective measures in place to detect potential security breaches.

How to build a disaster recovery team?

Whether creating a disaster recovery strategy from scratch or improving an existing plan, assembling the right collaborative team of experts is a critical first step. It starts with tapping IT specialists and other key individuals to provide leadership over the following key areas in the event of a disaster:

- **Crisis management:** This leadership role commences recovery plans, coordinates efforts throughout the recovery process, and resolves problems or delays that emerge.

- **Business continuity:** The expert overseeing this ensures that the recovery plan aligns with the company's business needs, based on the business impact analysis.

- **Impact assessment and recovery:** The team responsible for this area of recovery has technical expertise in IT infrastructure including servers, storage, databases and networks.
- **IT applications:** This role monitors which application activities should be implemented based on a restorative plan. Tasks include application integrations, application settings and configuration, and data consistency.

While not necessarily part of the IT department, the following roles should also be assigned to any disaster recovery plan:

- **Executive management:** The executive team will need to approve the strategy, policies and budget related to the disaster recovery plan, plus provide input if obstacles arise.
- **Critical business units:** A representative from each business unit will ideally provide feedback on disaster recovery planning so that their specific concerns are addressed.

How to plan for COVID-19 disaster recovery & business continuity

COVID-19 and the resulting global crisis have pushed many companies to support employees working remotely and forced organizations to rethink their disaster recovery and business continuity strategies. With the pandemic in play, even just a network outage can have a significant effect on the business.

Here are a few things to consider:

- Add the risks and potential consequences of infectious diseases to your disaster recovery plan. Although rare on such a global scale, having specific plans for this type of emergency will help ensure they're handled as smoothly as possible.
- Make plans for people, not just technology. The results of COVID-19 have shown that for businesses to remain successful employees need support, communication and resources. Plan ways that you will be able to provide these elements even when employees are working from home and may have different or limited access to their normal devices, networks or communication channels.
- Consider additional cloud and software-as-a-service (SaaS) solutions for more efficient and flexible options for remote work, as well as lessening the reliance on one central data center or main HQ. Make sure your plans include IT redundancy—multiple systems in multiple sites, so that if one system gets compromised, the business remains operational.

What are the benefits of disaster recovery software?

No organization can afford to ignore disaster recovery. The two most important benefits of having a disaster plan in place, including effective DR software, are:

- **Cost savings:** Planning for potential disruptive events can save businesses hundreds of thousands of dollars and even mean the difference between a company surviving a natural disaster or folding.
- **Faster recovery:** Depending on the disaster recovery strategy and the types of disaster recovery tools used, businesses can get up and running much faster after a disaster, or even continue operations as if nothing had happened.

6.11.2 Potential solutions and tradeoffs

At a high level, availability zones as well as multicloud architectures are potential solutions. We will address the multicloud topic in a future course.

We quote again from [18]:

What are the types of disaster recovery?

Businesses can choose from a variety of disaster recovery methods, or combine several:

- **Back-up:** This is the simplest type of disaster recovery and entails storing data off site or on a removable drive. However, just backing up data provides only minimal business continuity help, as the IT infrastructure itself is not backed up.
- **Cold Site:** In this type of disaster recovery, an organization sets up a basic infrastructure in a second, rarely used facility that provides a place for employees to work after a natural disaster or fire. It can help with business continuity because business operations can continue, but it does not provide a way to protect or recover important data, so a cold site must be combined with other methods of disaster recovery.
- **Hot Site:** A hot site maintains up-to-date copies of data at all times. Hot sites are time-consuming to set up and more expensive than cold sites, but they dramatically reduce down time.
- **Disaster Recovery as a Service** (DRaaS): In the event of a disaster or ransomware attack, a DRaaS provider moves an organization's computer processing to its own cloud infrastructure, allowing a business to continue operations seamlessly from the vendor's location, even if an organization's servers are down. DRaaS plans are available through either subscription or pay-per-use models. There are pros and cons to choosing a local DRaaS provider: latency will be lower after transferring to DRaaS servers that are closer to an organization's location, but in the event of a widespread natural disaster, a DRaaS that is nearby may be affected by the same disaster.
- **Back Up as a Service:** Similar to backing up data at a remote location, with Back Up as a Service, a third party provider backs up an organization's data, but not its IT infrastructure.
- **Datacenter disaster recovery:** The physical elements of a data center can protect data and contribute to faster disaster recovery in certain types of disasters. For instance, fire suppression tools will help data and computer equipment survive a fire. A backup power source will help businesses sail through power outages without grinding operations to a halt. Of course, none of these physical disaster recovery tools will help in the event of a cyber attack.
- **Virtualization:** Organizations can back up certain operations and data or even a working replica of an organization's entire computing environment on off-site virtual machines that are unaffected by physical disasters. Using virtualization as part of a disaster recovery plan also allows businesses to automate some disaster recovery processes, bringing everything back online faster. For virtualization to be an effective disaster recovery tool, frequent transfer of data and workloads is essential, as is good communication within the IT team about how many virtual machines are operating within an organization.
- **Point-in-time copies:** Point-in-time copies, also known as point-in-time snapshots, make a copy of the entire database at a given time. Data can be restored

from this back-up, but only if the copy is stored off site or on a virtual machine that is unaffected by the disaster.

- **Instant recovery:** Instant recovery is similar to point-in-time copies, except that instead of copying a database, instant recovery takes a snapshot of an entire virtual machine.

6.11.2.1 Disaster recovery as a service

One of the potential solutions, disaster recovery as a service, (DRaaS) is further described in [19]. We quote:

Disaster recovery as a service(DRaaS) is a cloud computing service model that allows an organization to back up its data and IT infrastructure in a third party cloud computing environment and provide all the DR orchestration, all through a SaaS solution, to regain access and functionality to IT infrastructure after a disaster. The as-a-service model means that the organization itself doesn't have to own all the resources or handle all the management for disaster recovery, instead relying on the service provider.

Disaster recovery planning is critical to business continuity. Many disasters that have the potential to wreak havoc on an IT organization have become more frequent in recent years:

- Natural disasters such as hurricanes, floods, wildfires and earthquakes
- Equipment failures and power outages
- Cyberattacks

Using DRaaS to prepare for a disaster

True DRaaS mirrors a complete infrastructure in fail-safe mode on virtual servers, including compute, storage and networking functions. An organization can continue to run applications—it just runs them from the service provider's cloud or hybrid cloud environment instead of from the disaster-affected physical servers. This means recovery time after a disaster can be much faster, or even instantaneous. Once the physical servers are recovered or replaced, the processing and data is migrated back onto them. Customers may experience higher latency when their applications are running from the cloud instead of from an on-site server, but the total business cost of downtime can be very high, so it's imperative that the business can get back up and running.

How does disaster recovery as a service work?

DRaaS works by replicating and hosting servers in a third-party vendor's facilities versus in the physical location of the organization that owns the workload. The disaster recovery plan is executed on the third-party vendor's facilities in the event of a disaster that shuts down a customer's site. Organizations may purchase DRaaS plans through a traditional subscription model or a pay-per-use model that allows them to pay only when disaster strikes. As-a-service solutions vary in scope and cost—organizations should evaluate potential DRaaS providers according to their own unique needs and budget.

DRaaS can save organizations money by eliminating the need for provisioning and maintaining an organization's own off-site disaster recovery environment. However, organizations should evaluate and understand service level agreements. For instance, what happens to recovery times if both the provider and customer are affected by the same natural disaster, such as a large hurricane or earthquake. Different DRaaS providers have different policies on prioritizing which customers get help first in a large regional disaster or allowing customers to perform their own disaster recovery testing.

Disaster recovery as a service advantages

Many businesses with lean IT teams simply can't afford to take the time needed to research, implement and fully test disaster recovery plans. DRaaS takes the burden of planning for a disaster off of the organization and puts it into the hands of experts in disaster recovery. It can also be much more affordable than hosting your own disaster recovery infrastructure in a remote location with an IT staff standing by if disaster strikes. If a disaster doesn't happen, that expensive second infrastructure and staff never get used. Many DRaaS providers charge you only if you need their services. For many organizations, DRaaS is a helpful solution to a nagging problem.

Is disaster recovery as a service right for you?

Organizations may choose to hand over all or part of their disaster recovery planning to a DRaaS provider. There are many different disaster recovery as a service providers to choose from, with three main models:

Managed DRaaS: In a managed DRaaS model, a third party takes over all responsibility for disaster recovery. Choosing this option requires an organization to stay in close contact with their DRaaS provider to ensure that it stays up to date on all infrastructure, application and services changes. If you lack the expertise or time to manage your own disaster recovery, this may be the best option for you.

Assisted DRaaS: If you prefer to maintain responsibility for some aspects of your disaster recovery plan, or if you have unique or customized applications that might be challenging for a third party to take over, assisted DRaaS might be a better option. In this model, the service provider offers its expertise for optimizing disaster recovery procedures, but the customer is responsible for implementing some or all of the disaster recovery plan.

Self-service DRaaS: The least expensive option is self-service DRaaS, where the customer is responsible for the planning, testing and management of disaster recovery, and the customer hosts its own infrastructure backup on virtual machines in a remote location. Careful planning and testing are required to make sure that processing can fail over to the virtual servers instantly in the event of a disaster. This option is best for those who have experienced disaster recovery experts on staff.

Whichever of these models suits you, VMware has a solution. If you would drive your own DRaaS solution to your own target DR site, you can consider solutions like Site Recovery Manager and VMware vSphere Replication. If you want to drive your own to VMware Cloud on AWS, you can consider solutions like VMware Cloud Disaster Recovery and VMware Site Recovery. If you would like a service provider to assist you with DR, whether fully managed or self service, consider VMware Cloud Director Availability from one of our DRaaS Validate partners

DRaaS vs. backup as a service (BaaS)

With disaster recovery as a service, the service provider moves an organization's computer processing to its cloud infrastructure in the event of a disaster. This way, the business can continue to operate, even if the original IT infrastructure is totally destroyed or held hostage. This differs from backup as a service, where only the data, but not the ability to process the data, is duplicated by a third-party provider. Because BaaS is only protecting the data, and not the infrastructure, it is typically less expensive than DRaaS. BaaS can be a good solution for companies that need to archive data or records for legal reasons, but most organizations who use BaaS will want to combine it with another disaster recovery tool to ensure business continuity.

Planning for disaster and getting the help you need is something every business needs to consider. Whatever option you choose, a disaster recovery plan is essential for business continuity, and organizations are increasingly turning to DRaaS.

Amazon offers Disaster Recovery of On-Premises Applications to AWS, as described in [20].

6.12 Multi-cloud design

—— This section is a stub for this semester. ——

This is a problem for the solutions architect role we described in Section 6.1.2. Recall that this role works for the consumer actor.

We address this role in the advanced course.

6.13 Hybrid cloud design

—— This section is a stub for this semester. ——

—— We will cover this topic in detail in the spring 2024 semester. ——

This is a problem for the solutions architect role we described in Section 6.1.2. Recall that this role works for the consumer actor.

We address this role in the advanced course.

6.14 Problem Section

Problems on generic technical requirements

Problem 6.1. Requirements and constraints. Consider the manageability requirement. Mention as many constraints as possible (well, not more than 10, anyway) that are pertinent to this requirement.

Problem 6.2. Requirements and constraints. Consider the performance requirement. Mention as many constraints as possible (well, not more than 10, anyway) that are pertinent to this requirement.

Problem 6.3. Tradeoffs due to cost constraint. Consider the availability requirement and the *cost* constraint.

1. Mention a few design choices that increase availability.
2. State a cost constraint.
3. Suggest two potential design options and outline the tradeoffs made in each; evaluate their impact of the tradeoffs on the availability requirement.
4. State how an architect would choose an option.

Problem 6.4. Tradeoffs in SaaS design. You are architecting a SaaS service.

1. Mention a few (business? technical?) factors that threaten (i.e., decrease) availability of this service.
2. Mention a few options you have at your disposal to combat these threats.
3. Mention a few constraints you must take into account.
4. Suggest two potential design options and outline the tradeoffs made in each; evaluate their impact of the tradeoffs on the availability requirement.
5. State how you would choose an option.

Problem 6.5. Challenges. Performance never appears in a SLA. Mention a few challenges that are specific to

1. IaaS
2. PaaS
3. SaaS
4. FaaS

Problems on specific design topics

Problem 6.6. Tenant collocation. Consider two tenants A and B that are using a SaaS service S at a provider's datacenter. Service S is running on a single container.

1. Suppose that tenant requests use the public internet and enter the datacenter through a gateway. Describe where and how tenant identification can be done.

2. Suppose that tenant requests use a private network (e.g., Azure's expressRoute) and enter the datacenter through a gateway. Describe where and how tenant identification can be done.
3. Which scenario offers more options?

Problem 6.7. GCP migration process. Read Google's guide for migrating to GCP, found in `https://cloud.google.com/architecture/migration-to-gcp-assessing-and-disco`
The document focuses on the "Assessing and discovering your workloads" phase of the migration process.

In this problem, we'll simulate (at least four of the six) steps involved in this phase. The environment is your personal laptop.

1. Build an inventory of the apps in your laptop. (It does not have to be comprehensive.)
2. Catalog your apps according to their properties and dependencies.
3. (Optional) Train and educate yourself on Google Cloud.
4. (Optional) Build an experiment and proof of concept on Google Cloud.
5. Calculate the total cost of ownership (TCO) of the target environment.
6. Choose the app(s) that you want to migrate first.

Problem 6.8. Migrating datasets. Read Google's guide for migrating (large) datasets, found in [22].

1. List some of the reasons that make transferring data to the cloud challenging. Rank these reasons.
2. Estimate the $ cost of the transfer process. State your assumptions clearly.
3. Estimate the time overhead of the transfer process. State your assumptions clearly.
4. Discuss the option offered by the "Transfer Appliance". When does it make sense to use it?

Problem 6.9. Cloud Networking - Connecting tenants to data centers. In Section 6.1.3, we mentioned that GCP offers *Dedicated Interconnect* as a VPN to connect consumer traffic to the GCP datacenters.

Suppose that your workloads running in Google Cloud require low latency between their virtual machine (VM) instances in a specified region and the Dedicated Interconnect colocation facility that you choose. In its documentation, Google says that you can select a low-latency (i.e., less than 5 milliseconds) colocation facility location.

1. Your users are all located in RTP, NC. Can you make a selection? If yes, provide the name of the GCP datacenter.
2. Your users are located in Santa Clara, CA and Bozeman Montana. Can you make a selection? If yes, provide the names of the GCP datacenters.

(Hint: you must search the GCP documentation we mentioned in Section 6.1.3.)

Problem 6.10. Load distribution. Search for literature on Facebook's Katran load balancer.

1. List the technical requirements Katran satisfies.
2. Summarize how Katran operates.
3. Mention, if you find them or if you can deduce them, the tradeoffs made in Katran's design.
4. Evaluate Katran. You may find the material in `https://www.facebook.com/cultofkatran` irrelevant, even though entertaining...

Problem 6.11. Load balancing and Reverse Proxy. Search for literature on the concept of Reverse Proxy.

1. Describe what a Reverse Proxy does.
2. Explain clearly the difference between the functions of Load balancing and Reverse Proxying.
3. Can you have both in an implementation? Explain.

Problem 6.12. Load balancing and multicloud. Consider a multicloud environment.

1. Explain how Load balancing would work in this environment.
2. Identify similarities between the single and multicloud scenarios.
3. Identify differences between the single and multicloud scenarios.

Problem 6.13. Disaster recovery. In the text, we presented VMware's view. Their examples of what is a disaster differed from Amazon's. Read Amazon's view on disasters in [21].

1. Outline one common element between the two approaches.
2. Outline one different element.

Huh?

Problem 6.14. In search of beautiful architectures. The world, not just the cloud is full of well-architected artifacts. Let's find some in the space of buildings, using the tool of `maps.google.com`.

1. Find an example of a beautifully architected building in Paris, France. Comment on the criteria you used to label it beautiful.
2. Find an example of a beautifully architected building in Paris, Kentucky. Comment on the criteria you used to label it beautiful.
3. Well-architected artifacts are copied again and again - that's why AWS put the Well-architected framework out there in the first place. How many times did the Eiffel Tower architectural blueprint has been implemented?
4. Having a well-architected framework and following principles is necessary for coming up with a complex system - but not sufficient. Faithfully executing the architectural diagrams is necessary. What do you think went wrong with the building shown in Figure 6.8?

Fig. 6.8: well-architected, not-so-well-implemented.

References

References marked as Handout H6.x are required reading material; you will find them in the required handouts binder in moodle. The rest are used in some form in the text or problems.
Reference [3] contains excellent material on load balancing; a word of caution - it's written by an expert, has a ton of information on how, but not on why.
Reference [5] is a must for your project.
There is a ton of references (and published work) on load balancing. We'll go over references [3], [9], [10], [12], and [13] in much greater detail in the advanced course.

1. **Handout H6.1:** Nikolas Roman Herbst, Samuel Kounev, Ralf Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not", pp. 23-27, 10th International Conference on Autonomic Computing (ICAC '13)
   https://www.usenix.org/system/files/conference/icac13/icac13_herbst.pdf
2. **Handout H6.2:** Load Balancing, ECE577/CSC577 Notes.
3. **Handout H6.3:** Matt Klein, Introduction to modern network load balancing and proxying, 2017
   https://blog.envoyproxy.io/introduction-to-modern-network-load-balancing-and-proxying-a57f
4. **Reference R6.1:** Joe Hertvik, Service Availability: Calculations and Metrics, Five 9s, and Best Practices,
   https://www.bmc.com/blogs/service-availability-calculation-metrics/
5. **Reference R6.2:** John S. Hammond, Ralph L. Keeney, and Howard Raiffa, "Even Swaps: A Rational Method for Making Trade-offs", https://hbr.org/1998/03/even-swaps-a-rational-method-for-making-trade-offs
6. **Reference R6.3:** Amazon Virtual Private Cloud (Amazon VPC), User Guide,
   https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html
7. **Reference R6.4:** What's AWS VPC? Amazon Virtual Private Cloud Explained,
   https://www.bmc.com/blogs/aws-vpc-virtual-private-cloud/

8. **Reference R6.5:** GPC VPC network overview, https://cloud.google.com/vpc/docs/vpc
9. **Reference R6.6:** Cooper Bethea et al, Managing Load https://sre.google/workbook/managing-load/
10. **Reference R6.7:** Eisenbud, Daniel E., et al. "Maglev: A fast and reliable software network load balancer." 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). 2016.
11. **Reference R6.8:** Cloud Load Balancing documentation, https://cloud.google.com/load-balancing/docs#docs
12. **Reference R6.9:** Derek DeJonghe, Load Balancing in the Cloud: Practical Solutions with NGINX and AWS, https://books.google.com/books?id=2zbFuQEACAAJ
13. **Reference R6.10:** What Is DNS Load Balancing? https://www.nginx.com/resources/glossary/dns-load-balancing/
14. **Reference R6.11:** Limitations of DNS-based load balancing https://learn.akamai.com/en-us/webhelp/global-traffic-management/global-traffic-management-user-guide/GUID-8B3CB734-ED76-4829-B778-45824CDFBEA7.html
15. **Reference R6.12:** Open-sourcing Katran, a scalable network load balancer, https://engineering.fb.com/2018/05/22/open-source/open-sourcing-katran-a-scalable-network-load-balancer/
16. **Reference R6.13:** What is Cloud Orchestration? https://www.vmware.com/topics/glossary/content/cloud-orchestration.html
17. **Reference R6.14:** Bruce Basil Mathews, "Service Mesh for Mere Mortals", 2021.
18. **Reference R6.15:** What is Disaster Recovery? https://www.vmware.com/topics/glossary/content/disaster-recovery.html
19. **Reference R6.16:** What is disaster recovery as a service (DRaaS)? https://www.vmware.com/topics/glossary/content/disaster-recovery-service-draas.html
20. **Reference R6.17:** "Disaster Recovery of On Premises Applications to AWS", AWS Whitepaper, https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-of-on-premises-applications-to-aws/disaster-recovery-of-on-premises-applications-to-aws.pdf
21. **Reference R6.18:** What is disaster recovery? https://aws.amazon.com/what-is/disaster-recovery/
22. **Reference R6.19:** Migration to Google Cloud: Transferring your large datasets, https://cloud.google.com/architecture/migration-to-google-cloud-transferring-your-large-da
23. **Reference R6.20:** Application Migration, https://www.vmware.com/topics/glossary/content/application-migration.html
24. **Reference R6.21:** AWS Direct Connect, https://aws.amazon.com/directconnect/
25. **Reference R6.22:** Azure ExpressRoute, https://azure.microsoft.com/en-us/services/expressroute/
26. **Reference R6.23:** Dedicated Interconnect overview, https://cloud.google.com/network-connectivity/docs/interconnect/concepts/dedicated-overview
27. **Reference R6.24:** The OpenStack Public Cloud Passport, https://www.openstack.org/passport/
28. **Reference R6.25:** What is Cloud Elasticity? https://www.vmware.com/topics/glossary/content/cloud-elasticity.html

29. **Reference R6.26:** Top 10 Skills for Data Architects,
https://datacatchup.com/top-10-skills-for-data-architects/
30. **Reference R6.27:** Monolithic vs. Microservices: a guide to application architecture,
https://www.talend.com/resources/monolithic-architecture/

Appendix A
Glossary and topics

The site `https://www.vmware.com/topics/glossary.html` contains an abundance of descriptions of hundreds of terms and topics in cloud computing. Several of the descriptions in these notes come from this source.

Another site with a smaller number of topics (around 20) is `https://www.redhat.com/en/topics`.

Appendix B
Useful Sites

Ask students for additional sites they found useful.

Site	Good for?
https://stackoverflow.com/	Technical questions
https://aws.amazon.com/premiumsupport/knowledge-center/	Questions on AWS services
https://www.quora.com/	General questions

Table B.1: Sites for answering questions.

Appendix C
Cloud certifications

C.1 AWS certifications

C.1.1 About AWS Cloud Certifications

C.1.2 Certification levels

Foundational Six months of fundamental AWS Cloud and industry knowledge AWS Certified Cloud Practitioner.

Associate One year of experience solving problems and implementing solutions using the AWS Cloud

AWS Certified Solutions Architect - Associate.

AWS Certified SysOps Administrator - Associate

AWS Certified Developer - Associate.

Professional Two years of comprehensive experience designing, operating, and troubleshooting solutions using the AWS Cloud

AWS Certified Solutions Architect - Professional.

AWS Certified DevOps Engineer - Professional.

Specialty Technical AWS Cloud experience in the Specialty domain as specified in the exam guide

AWS Certified Advanced Networking - Specialty

AWS Certified Data Analytics - Specialty

AWS Certified Database - Specialty

AWS Certified Machine Learning - Specialty

C.1.3 Professional level

C.2 Azure certifications

C.2.1 About Azure Cloud Certifications

C.2.2 Certification levels

Four levels
Fundamentals
MCSD
Role-based
Specialty
Six Roles
Administrator
Data Engineer
Developer
DevOps Engineer
Functional Consultant
Solution Architect
29 different certifications and 50 different exams
5 different certifications and 9 different exams for the Solution Architect

C.2.3 Professional level

C.3 Google certifications

C.3.1 About Google Cloud Certifications

we quote from `https://support.google.com/cloud-certification/answer/9750149?hl=en&ref_topic=9433215`

Google Cloud's role-based certifications measure an individual's proficiency at performing a specific job role using Google Cloud technology. The knowledge, skills, and abilities required for each job role are assessed using rigorously developed industry-standard methods. Google Cloud certifications empower individuals to advance their careers, and give employers the confidence to build highly skilled, effective teams.

Every Google Cloud certification starts with a formal study known as a Job Task Analysis (JTA). During the JTA, a panel of subject matter experts identifies the knowledge, skills, abilities, and experience required of an individual to perform the

tasks in a specific job role, and creates a detailed job description. These skills are described in the exam-specific exam guides and job descriptions that we publish on our website.

An exam guide provides the blueprint for developing exam questions and offers guidance to candidates studying for an exam. While we encourage candidates to be prepared to answer questions on any topic in an exam guide, we do not guarantee that every topic within an exam guide will be assessed.

After a candidate passes an exam and the candidate's exam performance is verified, a digital badge and certificate are issued. Each certificate contains a sequential number that reflects the candidate's position among the growing list of Google Cloud certified individuals.

Unless explicitly stated in the detailed exam descriptions, all Google Cloud certifications are valid for two years from the date certified. Candidates must recertify to maintain their certification status. Renewal notifications are sent 90, 60, and 30 days prior to the certification expiration date.

C.3.2 Certification levels

Google offers three levels of certifications, per `https://cloud.google.com/certification`

Foundational certification The foundational level certification validates broad knowledge of cloud concepts and Google Cloud products, services, tools, features, benefits, and use cases, specifically understanding the capabilities of Google Cloud.

This certification is appropriate for individuals in non-technical job roles who can add value in their organization by gaining Cloud knowledge and who have little or no hands-on experience in Google Cloud.

Associate certification The associate level certification is focused on the fundamental skills of deploying, monitoring, and maintaining projects on Google Cloud.

This certification is a good starting point for those new to cloud and can be used as a path to professional level certifications.

Professional certification Professional certifications span key technical job functions and assess advanced skills in design, implementation, and management.

These certifications are recommended for individuals with industry experience and familiarity with Google Cloud products and solutions.

C.3.3 Professional level

Recommended experience: 3+ years industry experience, including 1+ years on Google Cloud.

This level has exams on the following roles:

Cloud Architect

Cloud Developer
Data Engineer
Cloud Security Engineer
Cloud Network Engineer
Cloud DevOps Engineer
Machine Learning Engineer
Collaboration Engineer

C.3.3.1 Professional Cloud Architect

`https://cloud.google.com/certification/cloud-architect`

Professional Cloud Architects enable organizations to leverage Google Cloud technologies. With a thorough understanding of cloud architecture and Google Cloud, they design, develop, and manage robust, secure, scalable, highly available, and dynamic solutions to drive business objectives.

The Professional Cloud Architect certification exam assesses your ability to:

Design and plan a cloud solution architecture
Manage and provision the cloud solution infrastructure
Design for security and compliance
Analyze and optimize technical and business processes
Manage implementations of cloud architecture
Ensure solution and operations reliability

Cloud Architect
Cloud Developer
Data Engineer
Cloud Security Engineer
Cloud Network Engineer
Cloud DevOps Engineer
Machine Learning Engineer
Collaboration Engineer

C.4 Linux Foundation certifications

Check `https://training.linuxfoundation.org/certification/certified-kubernetes-application-developer-ckad/`

Index