Distributed Systems: Introduction (CO3404)

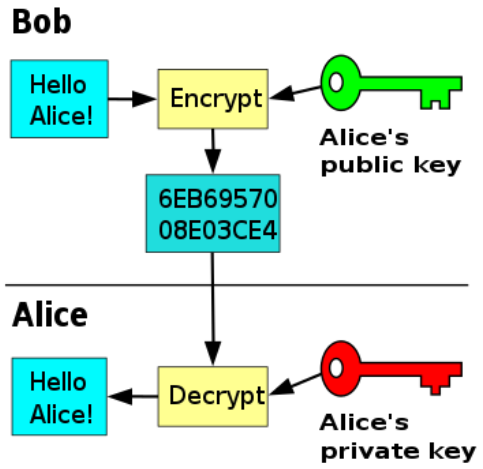
Dr Tony Nicol BSc (hons), MSc, PhD, MIET, CEng, FHEA, FBCS, CITP

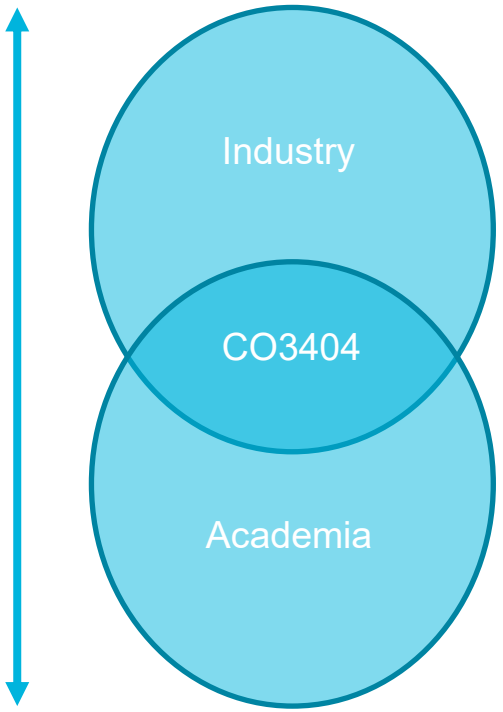
DWP: Head of Engineering

UCLAN: Senior Lecturer in Computer Science

My background

- Working in Industry and academia for 50 years. A few examples:
- Royal Marines Commando
- Marine Research: sonar system for nuclear Subs – Electronics /Software Engineer
- Naval Systems: sonar system for Trident class sub – Electronics / Software Engineer
- Crypto: Crypto systems. Hardware, Software, Firmware
- UCLAN: Electronics Engineering, Computing, Software Engineering Course Leader
- University of Liverpool: taught MSc in Computer Security & Cyber Forensics
- DWP: Solution Architect, Technical Architect, Data Architect, Deputy Director
- Back at UCLAN





The value of this module

- Enable development of current industry best practice skills and knowledge
- Full blown distributed systems are mainly used in large enterprises – i.e. big and complex
- Large organisations (enterprise) include Gov departments, large retailers such as Amazon, Asda, Banks
- Most large organisations are going through a technology and data transformation – you need to be part of that, and this module will ensure you are, as that's where the well-paid jobs are
- This module aims to give you a taste of the real world of IT so you stand out at interview and can be part of the next generation to develop enterprise systems with new technological approaches
- Distributed systems is a huge topic, so I'll concentrate on the current & future and not dwell on the past

Job prospects

1 to 25 of 755 Microservices Jobs in England

Java Software Engineer

England, United Kingdom

VASS UK&I

a team. Excellent communication and collaboration skills. Preferred Qualifications: Experience with cloud platforms (e.g., AWS, Azure) and containerization (e.g., Docker, Kubernetes). Knowledge of **microservices** architecture and RESTful API development. Familiarity with CI/CD pipelines and DevOps practices. Experience with test-driven development (TDD) and automated testing frameworks. VASS more »

Posted: 4 days ago

Senior Software Development Engineer

Greater London, England, United Kingdom

PCCW GLOBAL Limited

object-oriented design, SOLID principles, distributed system design and software design patterns Experience in developing a multi-tier architecture system Previous exposure in SOA, **Microservices** or using API Management tools is an advantage Experience in using container technologies (e.g. Kubernetes, Docker, etc.) is an advantage Experience in using Cloud platform more »

Posted: Yesterday

Salary Guide

Microservices England

10th Percentile
£43,750

25th Percentile
£52,500

Median
£65,000

75th Percentile
£87,500

90th Percentile
£110,000

More Microservices insights »

Related

RESTful 314

Spring Boot 127

Node.js 388

NoSQL 231

Java 1,069

React 899

REST 335

Spring 140

OAuth 92

AWS 2,159

JavaScript 1,397

AngularJS 528

PostgreSQL 445

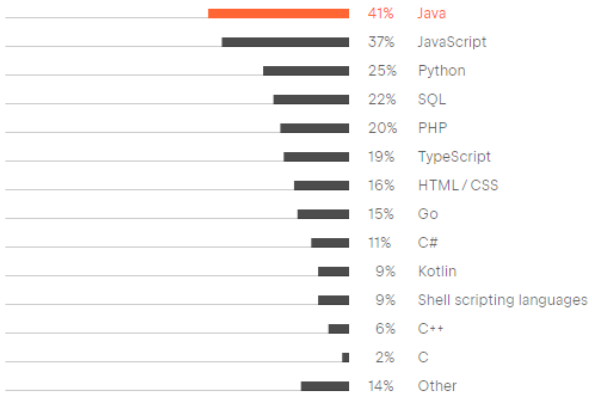
Serverless 394

Kafka 269

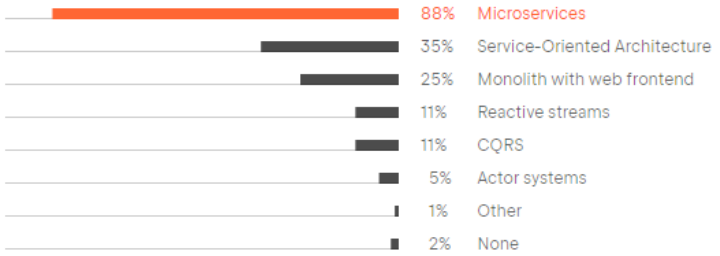
Relevant snippets

- There are other surveys of course …

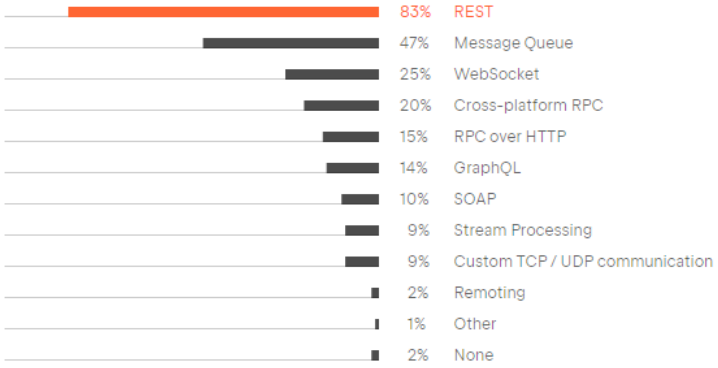
Primary languages among microservices developers



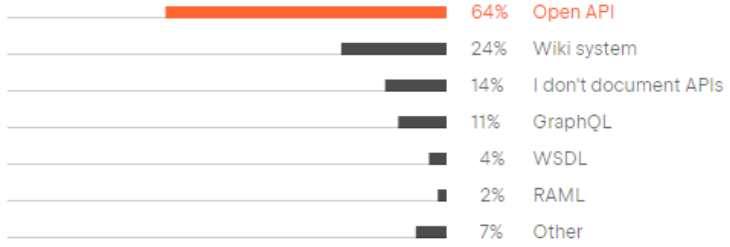
What approaches do you use in your system design?



How do the distributed parts of your application communicate?



How do you declare and document your APIs?



Module aims summary

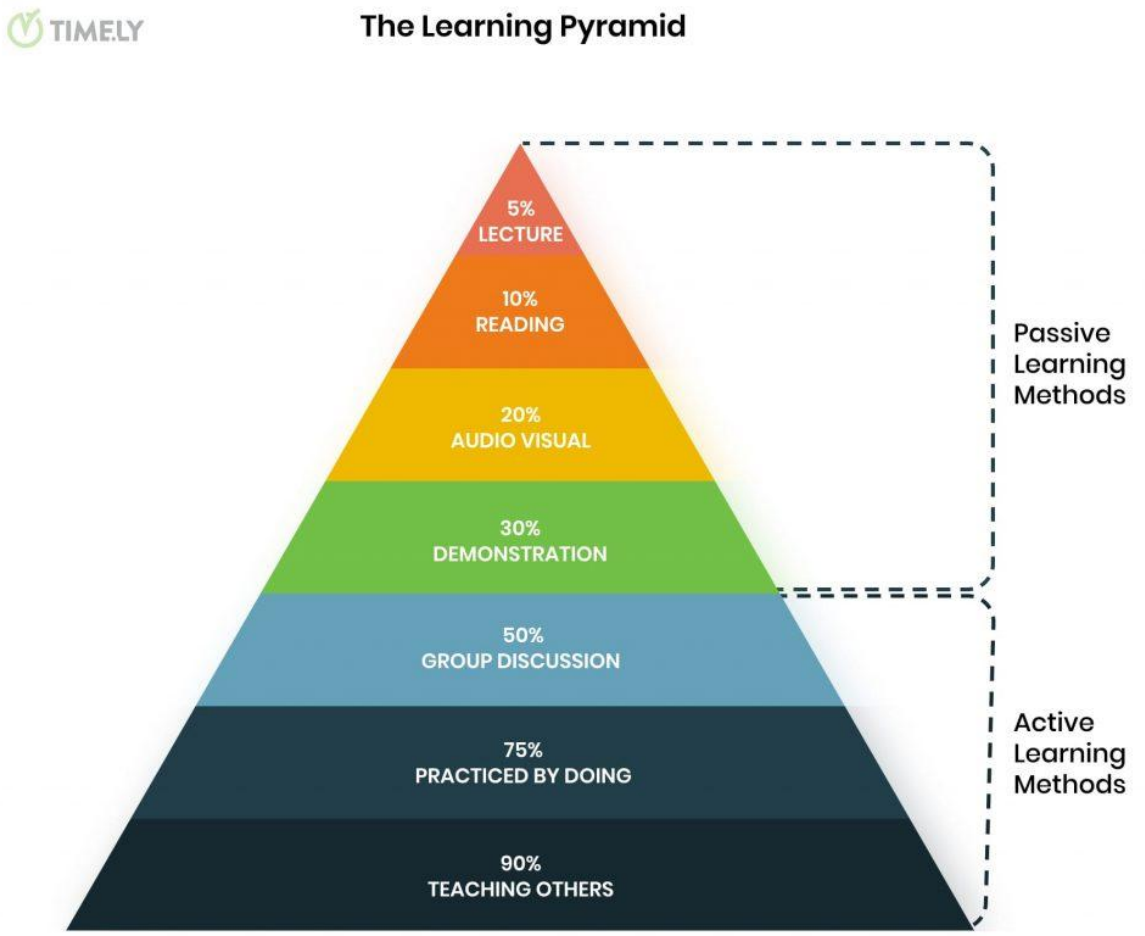
- 1. Create a distributed system application
  - Week by week I'll help you build on previous weeks leading to two assignments to create a distributed system
  - This will be built up practically with practical exercises were each week builds on the previous
- 2. Key design concepts such as scalability, security, heterogeneity, concurrency and containerisation.
- 3. Analyse technologies and patterns
- 4. How modern distributed systems support business
- 5. The need to deliver business value rather than technical perfection
- These 5 areas are a rough outline and don’t carry equal weight. We will spend a lot more time on the technology, but you do need to understand how it relates to business needs and constraints

- Assessment is in the form of a major assignment and an exam weighted at 50% and 50%

- My approach is: lecture 1h, demo/tutorial/Q&A 1h (2h alternate weeks) - drop-in labs
- May be lecture and demos in both depending on the relevance and timing

It is essential to practice!

- **Lectures** - are valuable but can't rely on them alone
- **Reading** - you need to re-read the slides as they are comprehensive. Research areas needing detail
- **Audio visual** - make sure you watch the videos I create. Don't think a 1 hour video takes one hour to watch - you will need to pause and research
- **Demonstrations** - will be provided most weeks illustrating practical implementation of concepts. You should also do the demos to move into active learning
- On the active learning side, this is learning by doing
- **Group discussion** - collaborative quizzes
- **Practice by doing** - you will have lab exercises - problem solving and implementing solutions will improve retention and understanding. This is an ESSENTIAL component of learning
- You will be given demonstration code to work with to provide a learning framework building the skills needed mainly for the assessed major assignment. Run study, understand and experiment with them
- **Teaching others** - interactive sessions. If answering questions in class is unbearable, tell me privately



Feedback from last year before we get started

- Positives
  - Some students did extremely well with grades in the 80s and two at 100% for coursework
  - "I got a job at the BBC directly because of this module"
  - "I didn't like the look of this module but now understand its importance in industry. It's now my favorite module"
  - "After discussing my microservices assignment I was offered a job at first interview"
  - "Having demos to work with to get an understanding of key concepts helped enormously with my assignment"
  - "This module directly helped me with my project. I changed the architecture to microservices"
  - You get the idea …

- The knowledge I gained from you during the Distributed Systems module was absolutely invaluable for my work in this team. My understanding of APIs, how they work, and how they run in the cloud was extremely helpful and it meant that I was able to start working on tickets and pushing code changes to production in my first week. Thank you again for making your module so insightful, engaging, and up to date with the industry.

Challenges

- A substantial number of students failed. My observations and feedback from students:
  - Missing classes. Attendance is monitored. If you get behind, tell me and we can work out how to catch-up
  - Underestimated the module. "I thought it was web development so expected to breeze through. That was a big mistake"
  - Poor planning. Spent too much time on other modules or project - need to distribute time effectively
    - Achievement coaches are available to talk to if you are struggling with planning, motivation, confidence building etc. ecanuti2@uclan.ac.uk
  - Listening but not hearing. "I'll watch the videos when I need to": procrastination is a very flawed strategy
    - Note the learning pyramid - you need continuous learning and practice to consolidate - binge learning doesn't work on challenging technical courses
  - Missing demos - these build the framework for the assignments. I build up week by week, you must practice then consolidate over the whole year
  - Not undertaking personal study - this is essential each week if you are to do well in the assignments - 152 hours is minimum
  - Not submitting assignment - yes - it's true. Several didn't. If you are struggling, you must ask me for help - don't be an ostrich
    - Student mentoring. Go to a session to discuss with student mentors. It looks good on your cv if you become a student mentor
  - Late submission. MCs are there to help but they can put you on the back foot so use sparingly
  - I don't need to know this stuff - I'm a C++ desktop developer. Good luck with that then. A degree won't get you a job - it may get you an interview
- Recent feedback
  - Don't focus on the web development and front-end JavaScript as core as it creates a false simplified impression of the module
  - Show an assignment up-front to make it clear how the different topics build up to it
  - Got too far behind for various reasons and didn't speak up to ask for help soon enough. (I will help if asked)
  - As lectures and demos recorded, I convinced myself I'd watch them when needed so focused on modules with less resources. That was a mistake

Assumptions

- You understand basic networking from Y1
- You understand basic database concepts such as normalisation, relationships - from Y1
- You can program - from Y1 and Y2
- You understand how to test code - from Y1 and Y2
- If this is not the case, e.g. you haven't studied the standard course from Y1 or have completely forgotten this stuff, then you need to tell me, and I'll cover in the tutorial or individually if necessary
- The HTLM and CSS we do is very basic so you should be able to pick it up in your own time - but I'm happy to go through these things and any other topic in the tutorial or drop-in labs
- You must tell me if you need help before it's too late. I' more than happy to help you get over mental blocks or clarify any of the concepts, but you must ask as I won't know
- I'll provide the lectures and tutorials / demos / Q&A online but this is only so you can join in the class to better see the code. If you have a valid reason not to be physically in the class, I need to be told ahead of time and I may allow you access from outside the class for exceptional circumstances

Responsibilities: Students

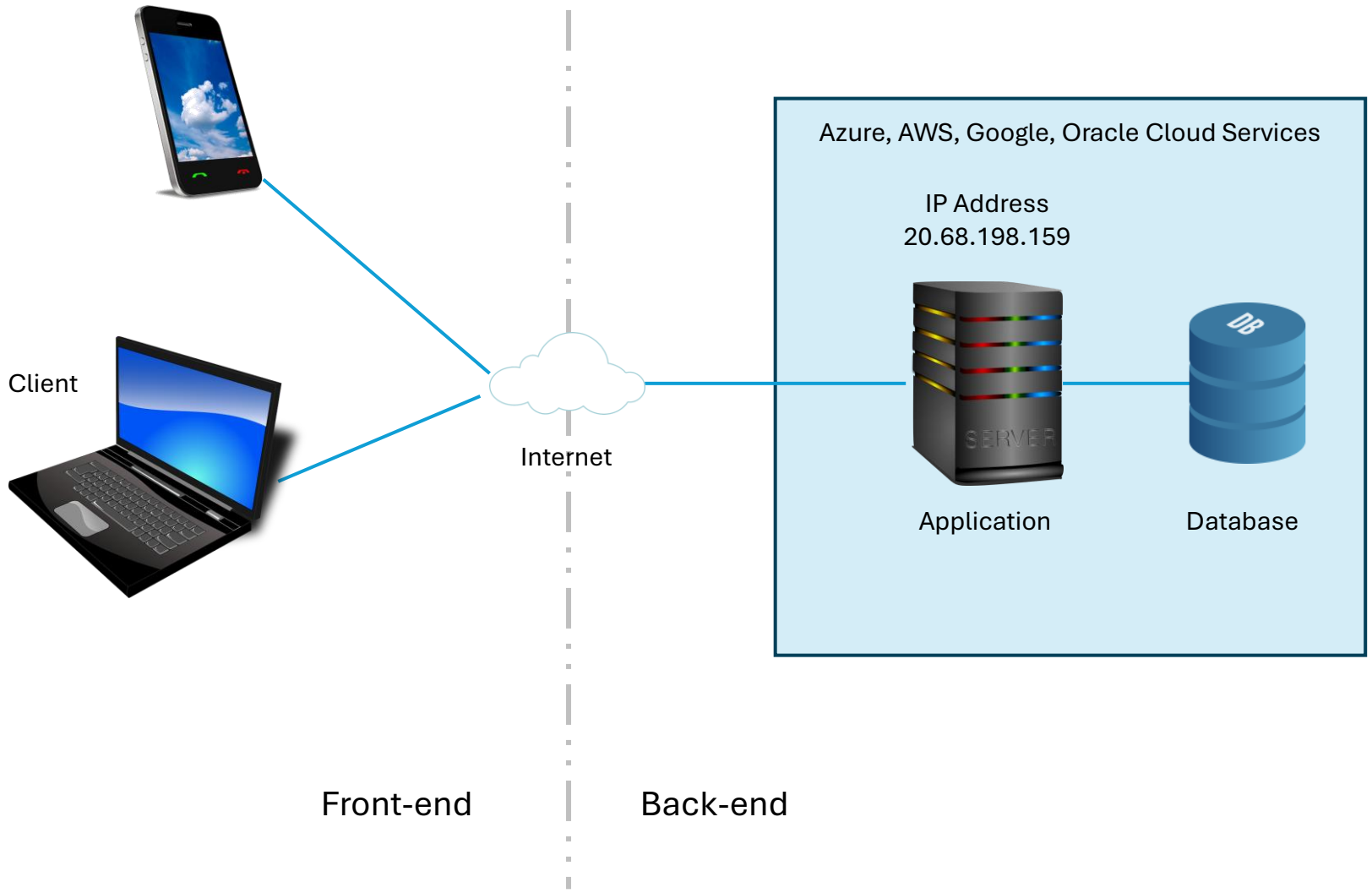
- Timeliness
  - Arrive in time to be prepared for the lecture/workshop
  - Manage your time for study effectively
- Communication
  - Talk to us for support, to raise issues and to seek advice/guidance
  - To monitor and use university communication mechanisms
- Participation
  - Attend all scheduled sessions and activities
  - To display professional conduct at all times
  - Contribute to group discussions and activities
- Personal study
  - Personal investigation of concepts and ideas
  - Prepare for assessments and exams
  - 152 hours minimum outside the module
- Behaviour
  - To treat all with respect: treat others as you would expect to be treated
  - To recognise and address/report all inappropriate behaviours and attitudes

Responsibilities: Ours

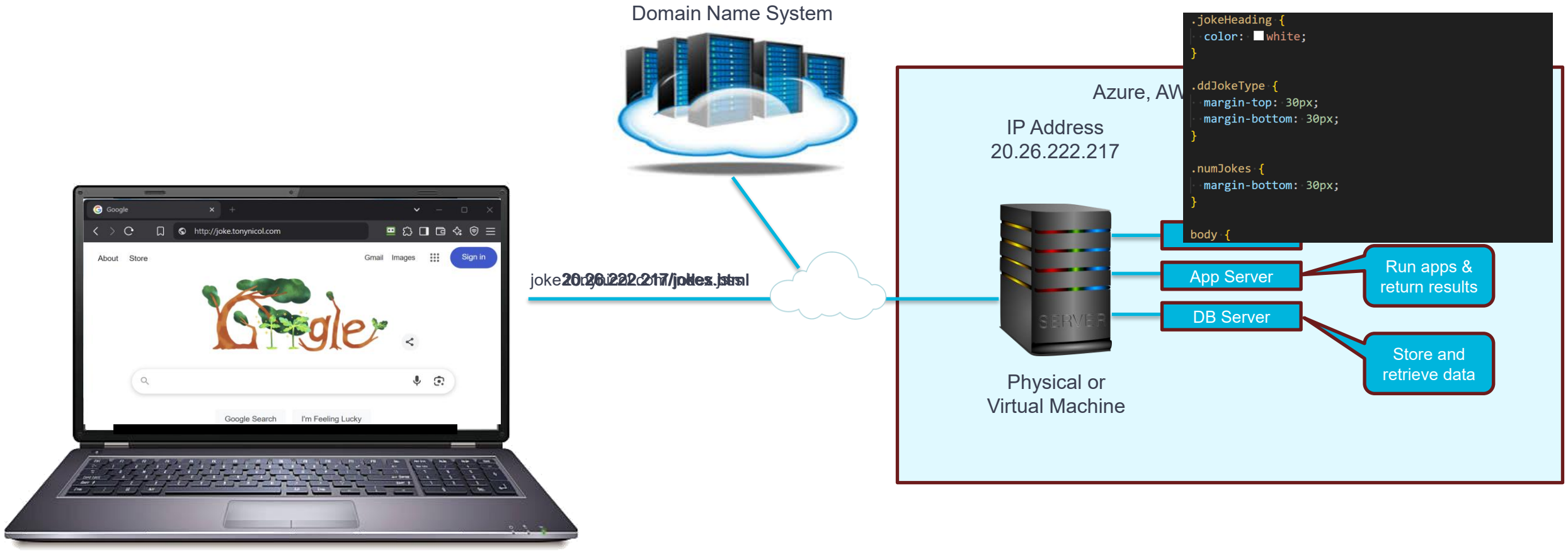
- Timeliness
  - Arrive in time to be prepared for delivery before the start of the lecture/workshop
  - To ensure all scheduled sessions and activities run effectively and as planned
- Communication
  - To provide clear communication of any changes to scheduled events and activities
  - To clearly communicate issues and adjustments that may affect study and assessment
  - To respond, in a timely manner, to issues raised
- Participation
  - To support and challenge learning and the development of understanding
  - To model and display professional conduct at all times
  - To always challenge inappropriate professional behaviours and attitudes whenever they occur
- Study
  - To professionally prepare and deliver high quality teaching and assessment
  - To fairly assess and report individual application of knowledge, skills and understanding
- Behaviour
  - To treat all with respect: treat others as you would expect to be treated
  - To recognise and address/report all inappropriate behaviours and attitudes

Basic 3-tier Client Server Architecture Monolithic

3-tier architecture



3-tier Architecture Demo



Demo joke site in debugger

Monolithic architecture

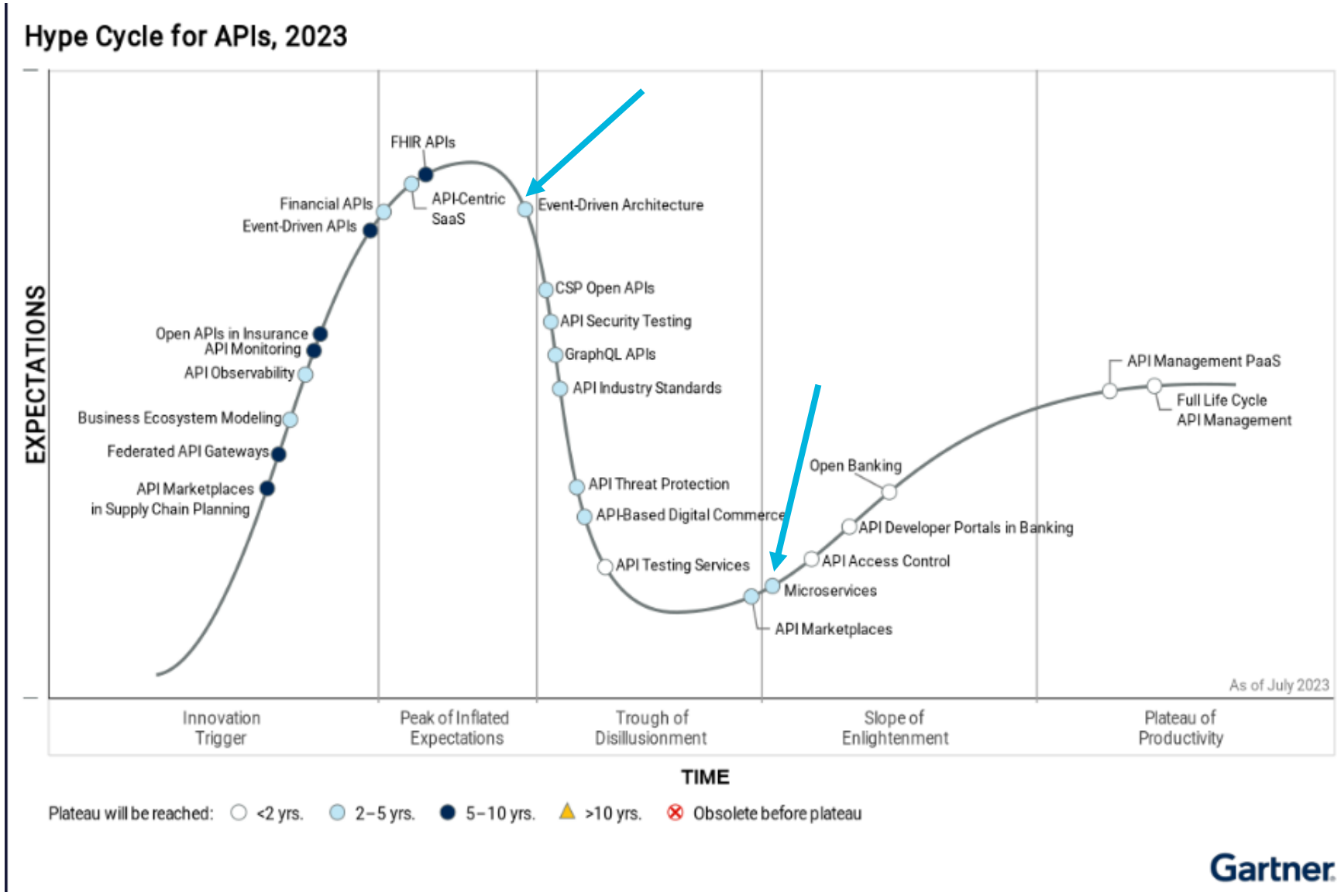
- Many if not most organisations implement a **monolithic** architecture which is basically one big program
- The application, even if it's designed to be modular, can still end up being effectively a single program with hundreds of thousands or millions of lines of code
- The application typically uses a single database which does make data management simpler, data consistency and performance is good
- The application may run on a cluster of servers for resilience, but as the cluster or server is increased to get more performance, the whole application needs to be replicated and scaled yet it may only be a small area of the application needing a performance improvement
- Code changes and testing can be extremely long winded and complicated because the applications are so large, complex, and business processes are tightly coupled - changing one area could create unintended side effects in another area so the whole application must be retested functionally and for performance. This is referred to as **Regression Testing**
- This level of testing can take months so organisations tend to save up problems and improvements leaving users to find work arounds until the next release - which could be months or years
- However, the monolith can be very performant through in-process function execution

- In many situations, a monolith probably the most suitable solution
- If the application is not particularly complex or large, e.g., a basic website for a small business, the monolithic approach like the one just shown, is perfectly reasonable
- The application is small enough for a person to understand and test so code changes won't take months
- However, large organisations such as Tesco, Amazon shopping, Netflix, Government departments etc., need to be able to adapt quickly to changes in business need or changes in government policy and they have huge user numbers of complex business processes
- Taking six months or more to make modifications to a business platform is no longer acceptable; a new architectural approach is needed

Distributed Systems

- A **distributed system** appears to be a single system but made up of multiple smaller systems
- When the smaller and simpler systems are used in combination, they appear to be a single complex system, but the underlying sub-systems are quicker to create and easier to maintain and test
- There are various architectural patterns such a service-oriented architecture (SOA), Event Driven Architecture, **Microservice** Architecture and others
- Currently the most popular approach today for distributed systems is a combination of event driven architecture and microservices to create an event-driven microservice architecture
- This is not a solution to all problems; each problem needs to consider the best architectural pattern which may well be a monolith or some other approach. This is something for solution and enterprise architects to ponder - engineers will design and implement the operational solution based on the architecture designed by the architects

Microservices



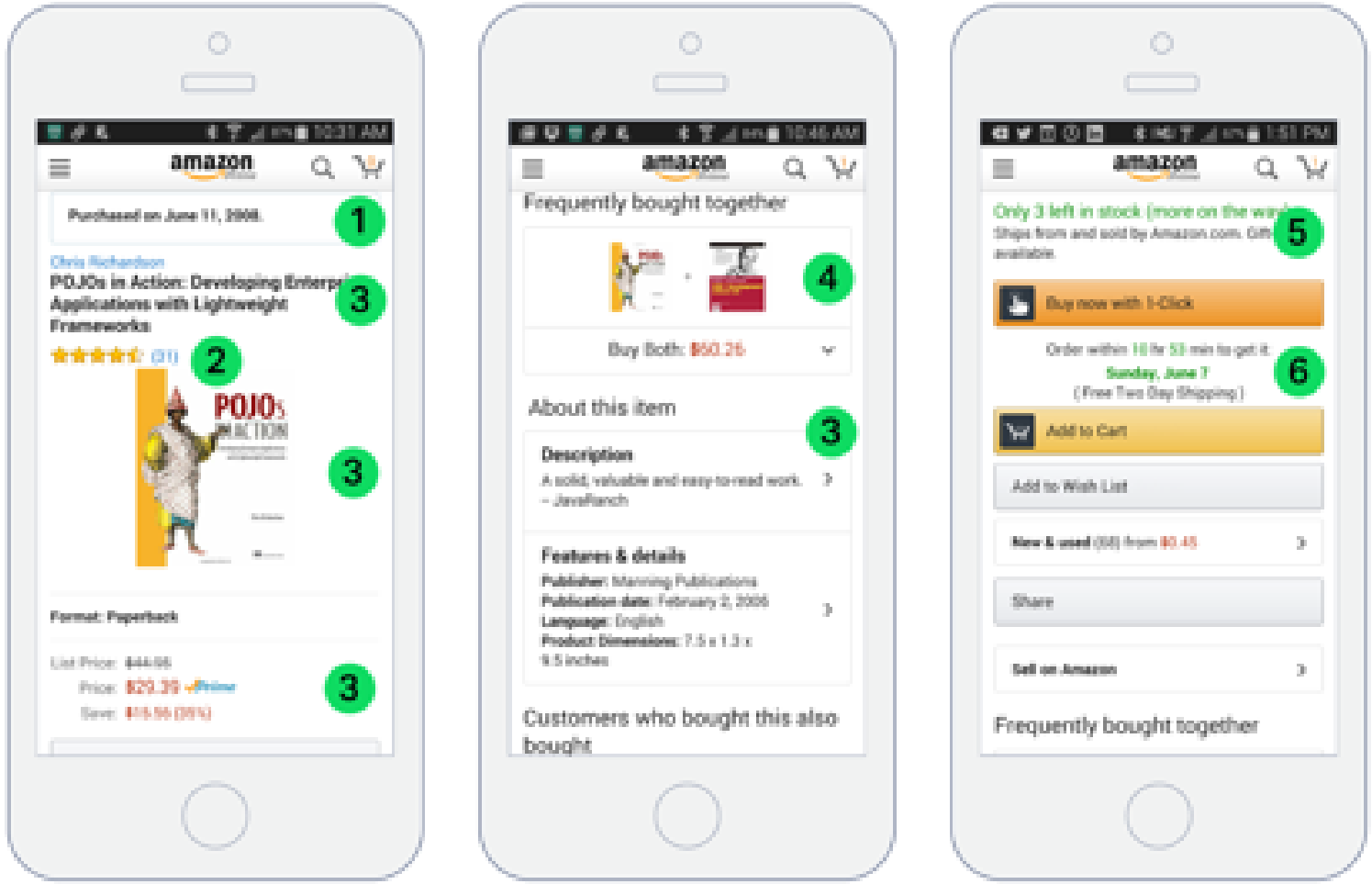
Microservices

- A microservice architecture aims to provide an agile response to continuous business change
- Rather than attempt to create one big application to solve a business problem, the problem is broken into business domain problems, potentially using domain driven design (DDD). i.e., individual areas of the business that can be implemented independently are identified
- This is similar to general software decomposition where functions or classes are identified, developed and tested in isolation then integrated into the final application. But, in a microservice architecture, the granularity of decomposition is far coarser, each component is a business service and is implemented as a complete application
- Common examples include:
  - Identity - login, registration etc
  - User profile - manage user details such as name, email, preferences etc
  - Order - manage order creation, status updates and history
  - Inventory - tracks stock levels, reservations and availability
  - Payment, notification, shipping and many more

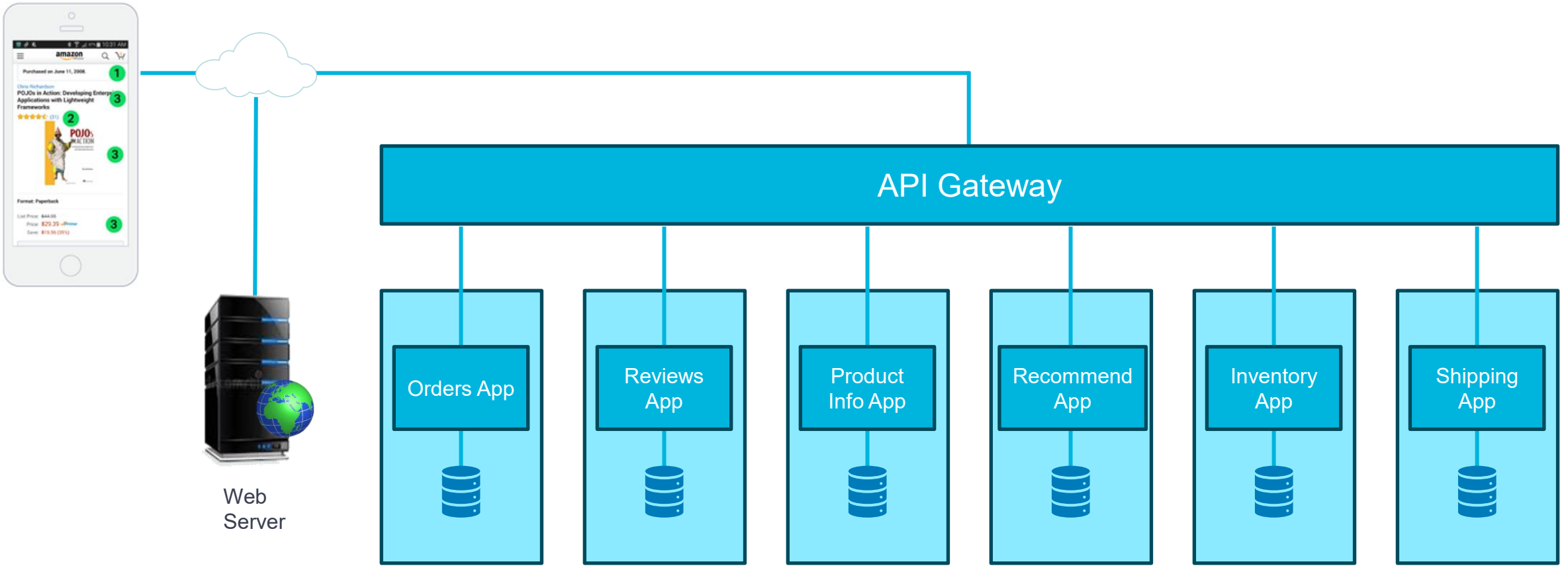
- A microservice:
  - is an independent application running on its own hardware with its own database if it needs one - databases aren't shared which prevents changes to data models from impacting other services
  - is loosely coupled to other microservices and a web front-end with well-defined and documented APIs
  - is typically developed, tested, deployed and maintained by a single multi-disciplinary team of 5-9 people including a product owner ("two pizza team")
  - is large enough to provide a business process, but small enough to enable daily or even hourly application updates
  - is agile in that the product pipeline generally used is fully automated CI/CD. e.g. at DWP, developers use DDD for the architecture and test-driven development (TDD) for the code. A lot of effort goes into the tests such that any code changes can be tested automatically and if successful, the code is deployed automatically to the target system
  - can be scaled independently. If one area of the business requires a performance improvement, that microservice can be scaled independently of the others so is more cost effective than the whole application scaling of a monolith
  - is resilient. If one microservice fails, that part of the business is unavailable, but the rest of the business can carry on
  - is a component of a heterogeneous architecture - as it is independent, the programming language, operating system, database technology, hardware etc can be whatever is the most appropriate for that business process

- Consider Amazon shopping. This is a large complex business - too large and complex for a monolith
- Independent business areas are identified and microservices implemented. At a high level, let's split that into six services:
  - Order history
  - Reviews
  - Product information
  - Recommendations
  - Inventory
  - Shipping
  - etc
- These six (there will be more than this) can be developed in isolation but once deployed, are available as a component making up the whole business service that is Amazon shopping
- To the user, it looks like one business as the front-end requests resources from various microservices to populate the web page

Example shopping site



1. ORDER HISTORY
2. REVIEWS
3. BASIC PRODUCT INFO
4. RECOMMENDATION
5. INVENTORY
6. SHIPPING


API Gateway
Orders App
Reviews App
Product Info App
Recommend App
Inventory App
Shipping App
Web Server

- Like all good ideas, there are challenges that need to be considered:
  - As components of the service interact across a network, latency is introduced - unlike the monolith
  - Data consistency is a challenge when data comes from various databases as opposed to one. e.g. for speed, you may want a local copy of some data, so managing consistency with the master can be a challenge
  - Fault finding can be more difficult as each team focusses on one area it can be difficult to track the problem through the whole system
  - Communication between microservices can become complex
  - It is a myth that microservices promote reusability or they are cheaper to build and operate
  - If your service does not require frequent or at least regular business change, then microservices are probably not appropriate as the key benefit is agility to business change
- So microservices are not a silver bullet that will kill the monolith, each have their advantages and disadvantages. So, like all design challenges, a decision as to which approach to use is based on the current and anticipated future business needs

- A Gartner definition of a microservice:
- Microservices are application components that are tightly scoped, highly cohesive, strongly encapsulated, loosely coupled, independently deployable and independently scalable. They are deployed as services behind open, standard network-accessible interfaces with well-defined API contracts. These components own the data they present, and only present data that they own

  [Gartner, Inc. | G00745207, page 33]

- A microservice architecture is an application architecture deployed as a distributed system

Business needs

- A system is there to satisfy a need which is typically a business need. i.e. we don't build a solution then ask what can we use it for, we take a business need and create a solution to satisfy that need
- The business need may be a new business line or an improvement of current business lines to **respond more quickly to changing customer needs and market trends**
- The business needs will be gathered by a business analyst and maybe a product owner, then passed to an architect. The architects (solution, network, data, application, enterprise etc) will provide the blueprint for the solution which is then implemented by software engineers, network engineers, security engineers, cloud engineers etc. None of these people do everything but need an awareness of it all
- The key point is that the business drives the IT not the other way round and it is a team task and not that of an individual so being able to work well in a multidisciplinary team is important as is having a working knowledge of what the other team members are specialists in; this module attempts to give you that
- There needs to be a business case for funding, technical options are presented and considered against costs and benefits. Considerations such as data protection, system resilience, performance, security, accessibility, operational cost, software licenses, maintenance, telemetry, operating model are all business considerations for any system so requirements tend to fall into two areas: functional and non-functional
  - Functional requirement: what a system **must do** e.g. "When I press the green button, the red light should light"
  - Non-functional requirement: **how well it does it**. e.g., "When the green button is pressed, the red light should light within 0.5s
- Both will be provided and assessed in the assignments

2024/25 Assignment

Public internet
Azure private network
Docker private network
Virtual machine
Application container
Rabbitmq combined container

80 / 443
kong
8445
public IP:
private IP:

/categories
/question/:category
/moderate
/categories
/categories
/submit
/docs

4000
question
3000
/categories

4100
moderate
3100
/categories
submitted question

4200
submit
3200
categories

4002
MongoDB
27017

4002
MySQL
3306
mysql
vol

categories

4102
rmq-admin
15672

4201
SUBMITTED_QUESTIONS
5672

4202
rmq-admin
15672

public IP:
private IP:

mongo
vol

4001
etl
3001

moderated questions

4101
MODERATED_QUESTIONS
5672

public IP:
private IP:

public IP:
private IP:

Container port number
Application port number

Development Approach

- I'll build up the knowledge of the components in the architecture week-by-week
- This does mean that successive weeks build on previous weeks, so missing classes is not advisable
- I'll discuss and provide demo code for most of the components
  - You need to work through them to familiarise yourself with the operation
  - You need to do the lab exercises to practice
  - If you don't, your assignments will be very difficult to get working