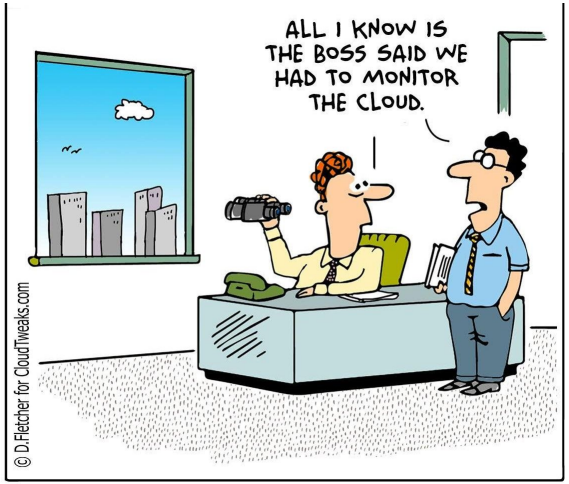
Cloud Computing & RFCs

99-520 Summer 25



1970s Teleprocessing

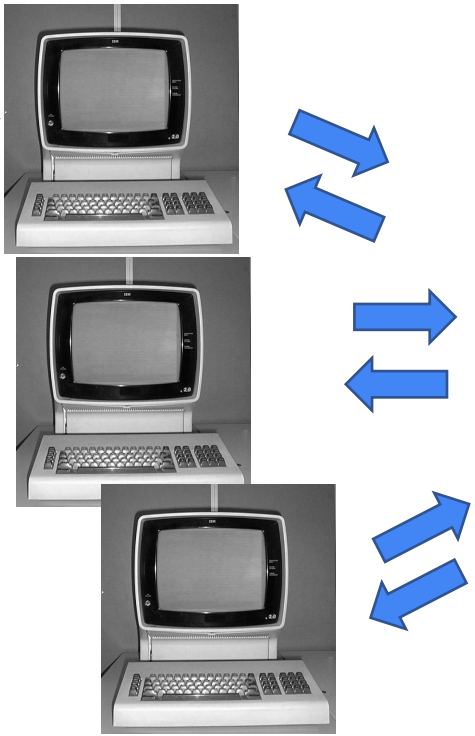


Photo Credit: University of

Photo Credit: ArnoldReinhold, CC BY-SA 3.0 via Wikimedia Commons

Photo Credit: Wikipedia

Credit: Jeff Eppinger (17-437)

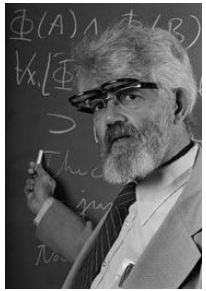
1980s & 1990s Personal Computing

Photo Credit: Rama & Musée Bolo, CC BY-SA 2.0 FR, via Wikimedia Commons

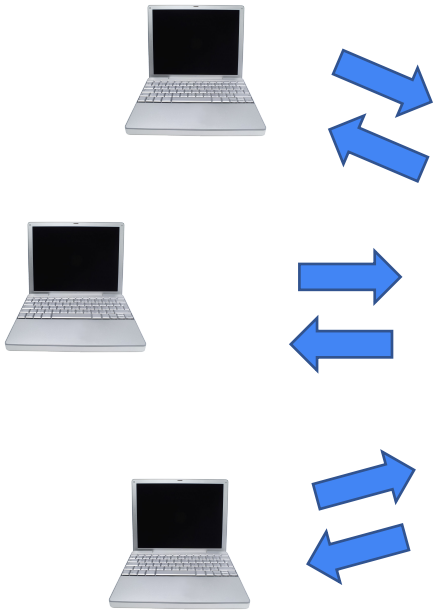
Photo Credit: Alexander Schaelss, CC BY-SA 3.0 via Wikimedia Commons

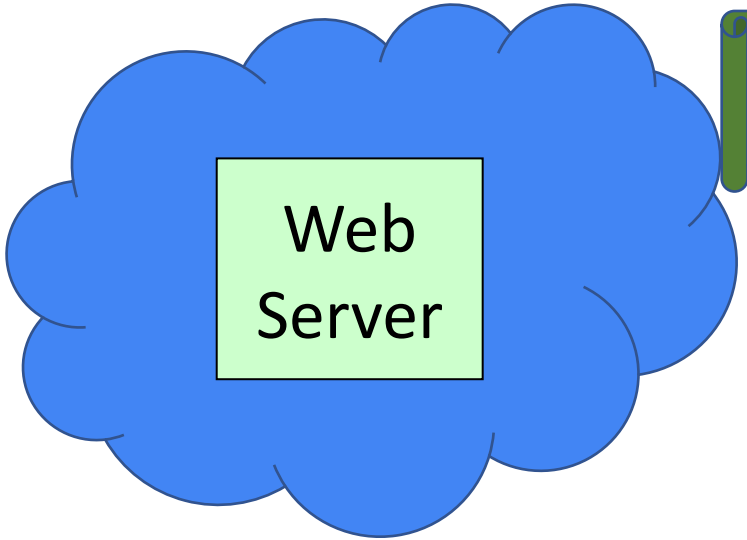
Credit: Jeff Eppinger (17-437)

2000s Cloud Computing



"Computing may someday be organized as a public utility just as the telephone system is a public utility…Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system …"



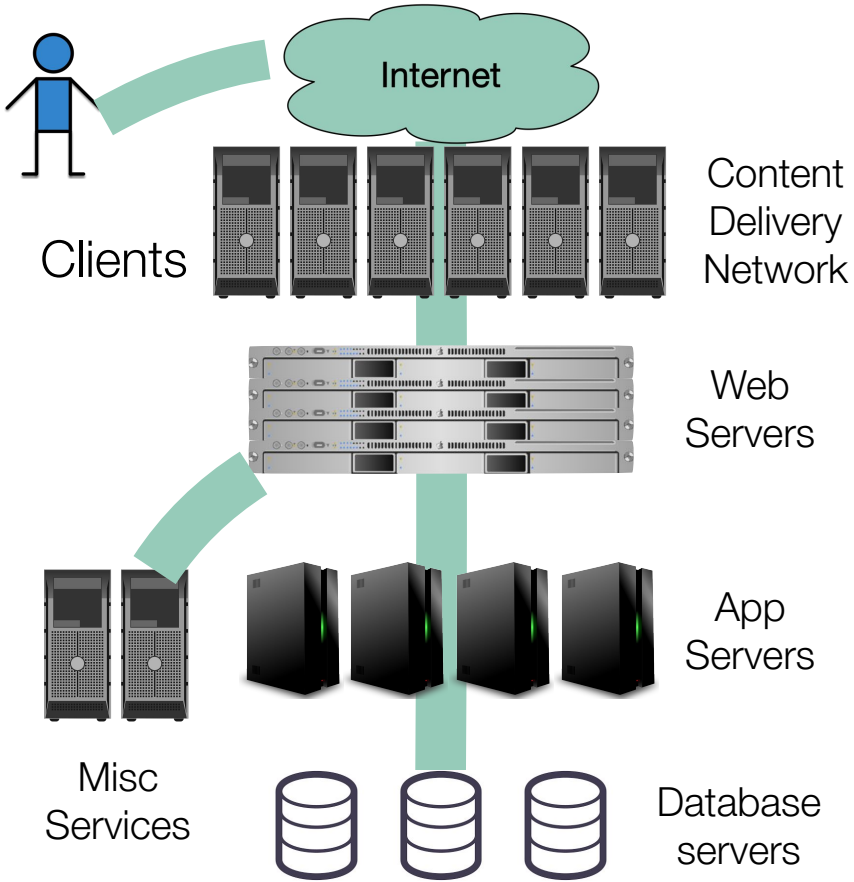


McCarthy's predictions come true!

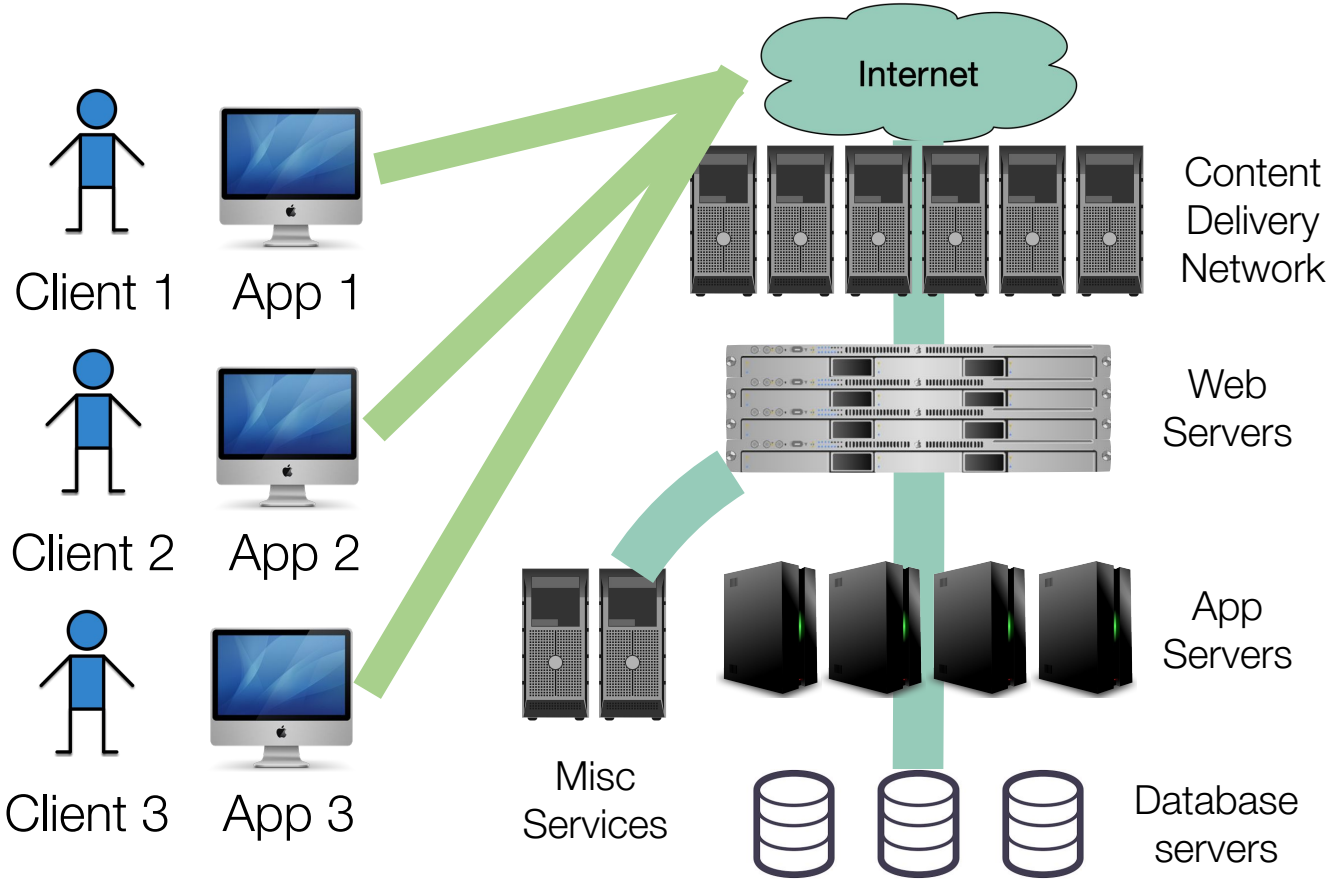
Credit: Jeff Eppinger (17-437)

Many apps rely on common infrastructure

- Content delivery network: caches static content “at the edge” (e.g. cloudflare, Akamai)
- Web servers: Speak HTTP, serve static content, load balance between app servers (e.g. haproxy, traefik)
- App servers: Runs our application (e.g. nodejs)
- Misc services: Logging, monitoring, firewall
- Database servers: Persistent data

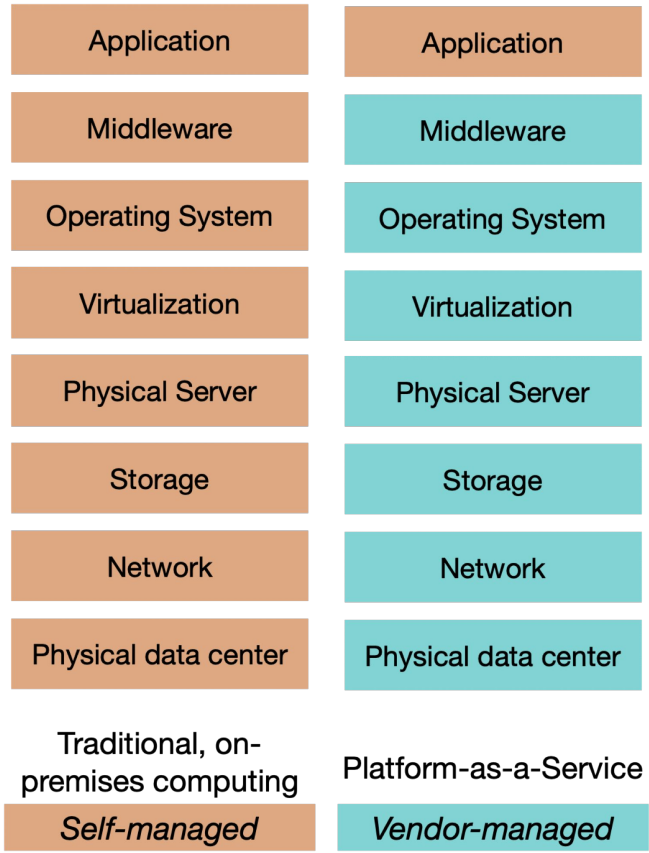


What elements can be shared?



What elements can be shared?

- Our apps run on a “tall stack” of dependencies
- Traditionally this full stack is self-managed
- Cloud providers offer products that manage parts of that stack for us:
  - “Infrastructure as a service”
  - “Platform as a service”
  - “Software as a Service”



Multi-Tenancy creates economies of scale

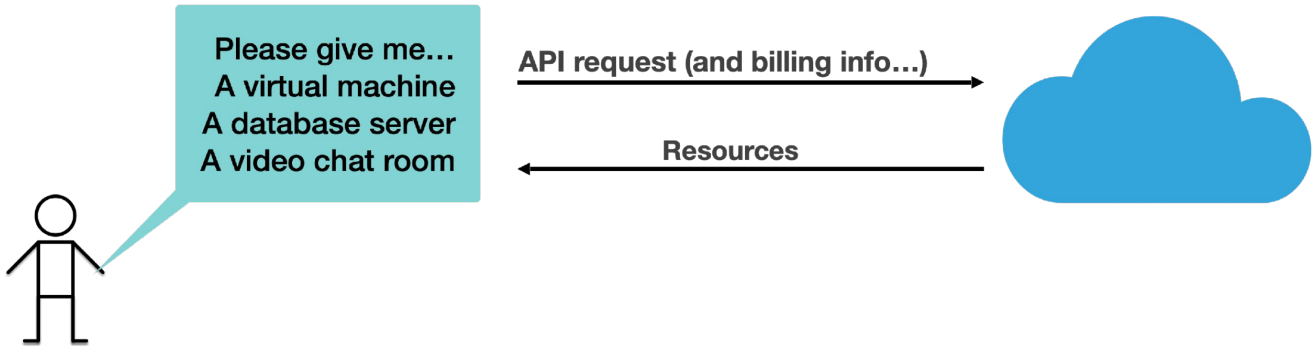
- At the physical level:
  - Multiple customers' physical machines in the same data center
  - Save on physical costs (centralize power, cooling, security, maintenance)
- At the physical server level:
  - Multiple customers' virtual machines in the same physical machine
  - Save on resource costs (utilize marginal computing capacity – CPUs, RAM, disk)
- At the application level:
  - Multiple customer's applications hosted in same virtual machine
  - Save on resource overhead (eliminate redundant infrastructure like OS)
- "Cloud" is the natural expansion of multi-tenancy at all levels

Cloud infrastructure scales elastically

- “Traditional” computing infrastructure requires capital investment
  - “Scaling up” means buying more hardware, or maintaining excess capacity for when scale is needed
  - “Scaling down” means selling hardware, or powering it off
- Cloud computing scales elastically:
  - “Scaling up” means allocating more shared resources
  - “Scaling down” means releasing resources into a pool
  - Billed on consumption (usually per-second, per-minute or per-hour)

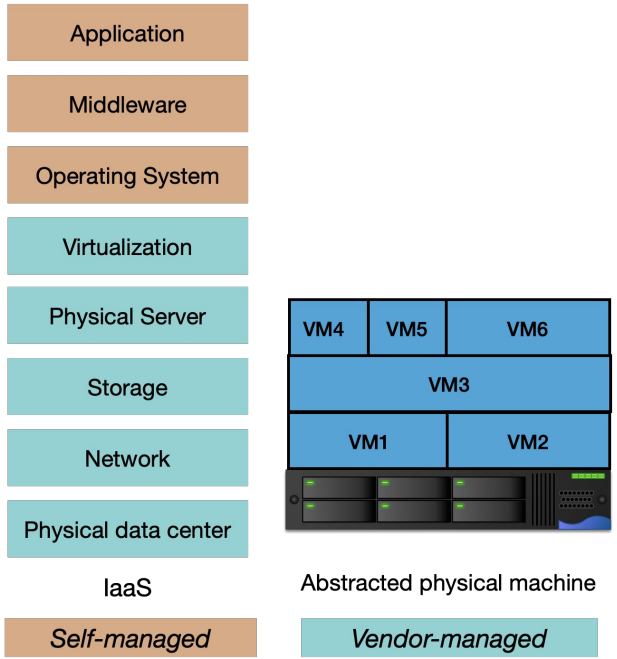
Cloud services gives on-demand access to infrastructure, “as a service”

- Vendor provides a service catalog of “***X as a service***” abstractions that provide infrastructure as a service
- APIs or web portals allow us to provision resources on-demand
- Transfers responsibility for managing the underlying infrastructure to a vendor



Infrastructure as a Service: Virtual Machines

- Virtual machines:
  - Virtualize a single large server into many smaller machines
  - Separates administration responsibilities for physical machine vs virtual machines
  - OS limits resource usage and guarantees quality per-VM
  - Each **VM runs its own OS**
  - Examples:
    - Cloud: Amazon EC2, Google Compute Engine, Azure
    - On-Premises: VMWare, Proxmox, OpenStack

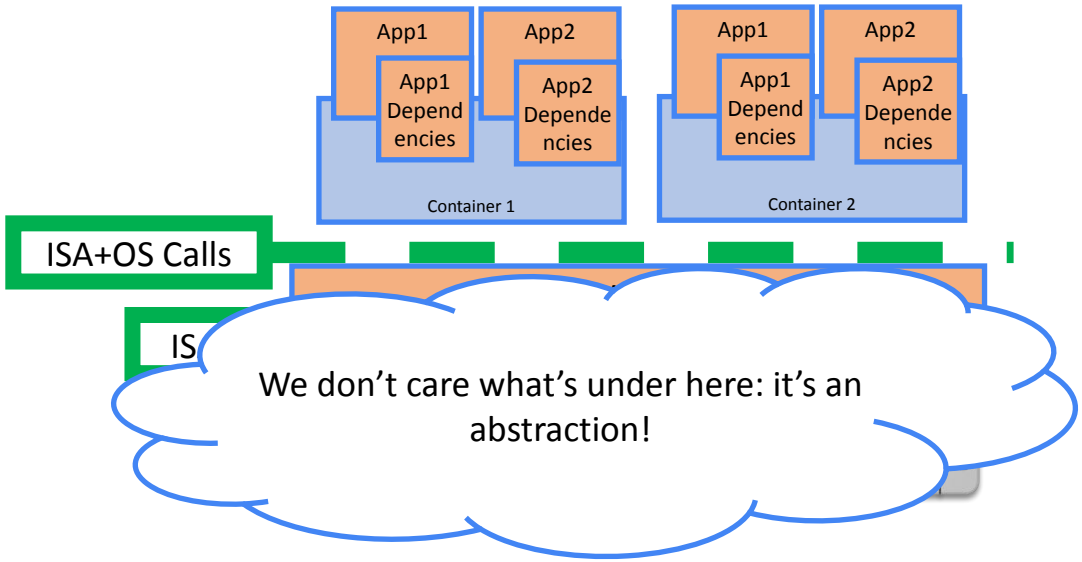


Virtual Machines to Containers

- Each VM contains a **full operating system**
- What if each application could run in the same (overall) operating system? Why have multiple copies?
- Advantages to smaller apps:
  - Faster to copy (and hence provision)
  - Consume less storage (base OS images are usually 3-10GB)

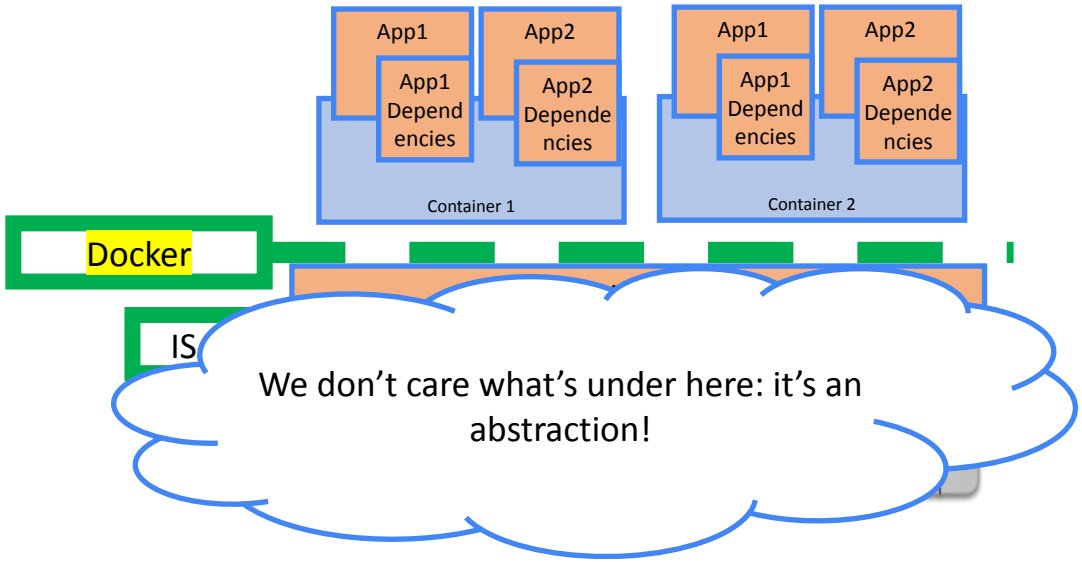
XaaS: Containers as a Service

- Vendor supplies an on-demand instance of an operating system
  - Eg: Linux version NN
- Vendor is free to implement that instance in a way that optimizes costs across many clients.



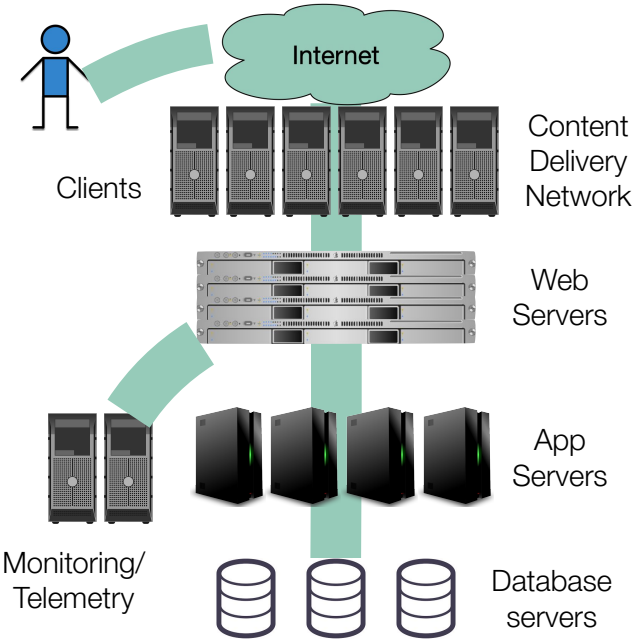
Docker is the prevailing container platform

- Docker provides a standardized interface for your container to use
- Many vendors will host your Docker container
- An open standard for containers also exists (“OCI”)



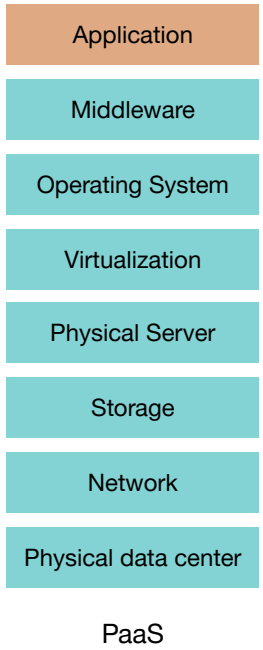
Platform-as-a-Service **(PaaS)**: vendor supplies OS + middleware

- Middleware is the stuff between our app and a user's requests:
  - Content delivery networks: Cache static content
  - Web Servers: route client requests to one of our app containers
  - Application server: run our handler functions in response to requests from load balancer
  - Monitoring/telemetry: log requests, response times and errors
- Cloud vendors provide managed middleware platforms too: "**Platform as a Service**"



PaaS is often the simplest choice for app deployment

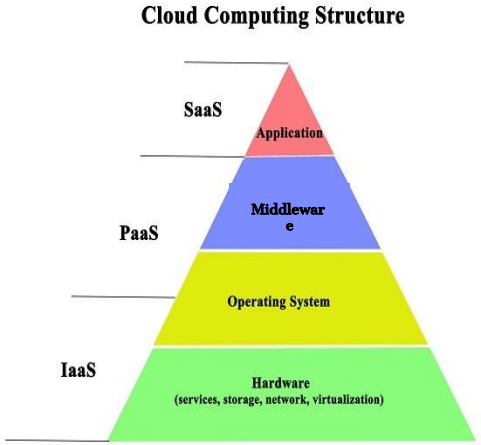
- **Platform-as-a-Service** provides components most apps need, fully managed by the vendor: load balancer, monitoring, application server
- Some PaaS run your app in a container: Heroku, AWS Elastic Beanstalk, Google App Engine, Railway, Vercel…
- Other PaaS run your apps as individual functions/event handlers: AWS Lambda, Google Cloud Functions, Azure Functions
- Other PaaSs provide databases and authentication, and run your functions/event handlers: Google Firebase, Back4App



Activity

Pick one **cloud service model** based on your team number

- Teams 1, 2: Software as a Service - SaaS
- Teams 3, 4, 5: Platform as a Service - PaaS
- Teams 6, 7: Infrastructure as a Service - IaaS



- Brainstorm and come up with one real-world scenario where the assigned cloud service model (IaaS, PaaS, or SaaS) would be the most convenient or optimal choice.
- Identify why their model is the best fit for the scenario and compare it briefly with the other two models to highlight the advantages of choosing their model.

Public Clouds are not the only choice

- "Public" clouds are connected to the internet and available to anyone.
  - Examples: Amazon, Azure, Google Cloud, DigitalOcean.
- "Private" clouds use cloud technologies with **on-premises** hardware and are self-managed.
  - They are cost-effective when a large scale of basic resources is needed.
  - Examples of management software: OpenStack, VMWare, Proxmox, Kubernetes.
- "Hybrid" clouds integrate private and public clouds
  - They are an effective option for handling capacity bursts from the private to the public cloud.

OpenStack

- OpenStack is an open-source software platform that enables the creation and management of private and public cloud infrastructures.
- It supports the provisioning of virtual machines, storage, networking, and other Infrastructure-as-a-Service (IaaS) components.
- Optimized for large-scale deployments with thousands of nodes, it is used in enterprise data centers and large private clouds.
- It is complex to set up and requires expertise to manage effectively.

PROXMOX

- Proxmox VE (Virtual Environment) is an open-source virtualization platform that combines the management of virtual machines (VMs) and containers.
- It is ideal for creating and managing private clouds and virtualization environments in data centers.
- Well-suited for small to medium-sized deployments, it can handle clusters but is best for moderately scaled environments such as small to mid-sized businesses or research labs.

CLOUD NATIVE ECOSYSTEM / INFRASTRUCTURE AS CODE / OPERATIONS

Why Companies Are Ditching the Cloud: The Rise of Cloud Repatriation

Major organizations like 37signals and GEICO highlight the economic and strategic reasons to reconsider cloud infrastructure.

Nov 5th, 2024 4:00am by Rob Pankow

37signals is completing its on-prem move, deleting its AWS account to save millions

Industry ‘pulled a fast one convincing everyone cloud is the only way’ says CTO David Heinemeier Hansson

Simon Sharwood

Fri 9 May 2025 // 06:39 UTC

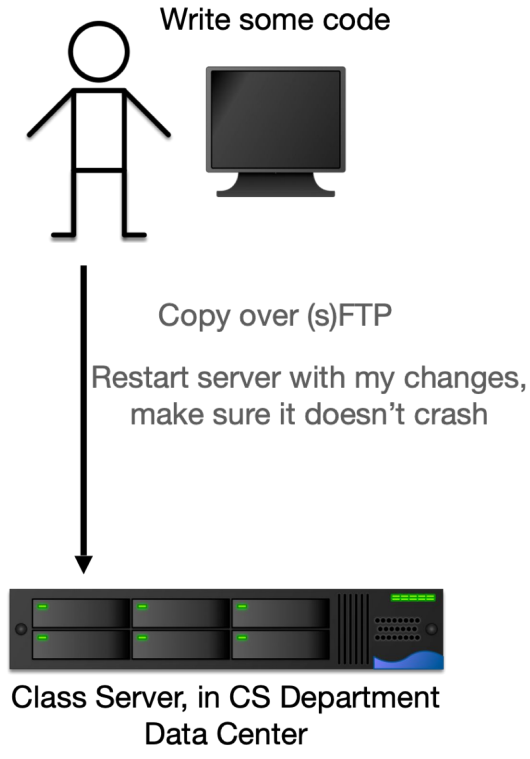
How to deploy web apps?

What we need:

- An application (codebase)
- A server that can run our application
- A network that is configured to route requests from an address to that server

Questions to think about:

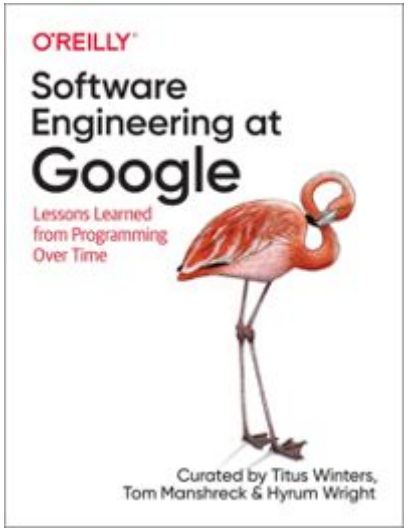
- What software do we need to run besides our application code? (Databases, caches, etc?)
- Where does this server come from? (Buy/Borrow?)
- Who else gets to use this server? (Multi-tenancy or exclusive?)
- Who maintains the server and software? (Updates OS, libraries, etc?)



RFCs
(Request For Comments)

Types of documentation

- Reference documentation (incl. code comments)
- Design documents
- Tutorials
- Conceptual documentation
- Landing pages



RFCs

- Code review before there is code!
- Collaborative (Google Docs)
- Ensure various concerns are covered, such as: security implications, internationalization, storage requirements, and privacy concerns.
- A good design doc should cover
  - Goals and use cases for the design
  - Implementation ideas
  - Propose key design decisions with an emphasis on their individual tradeoffs

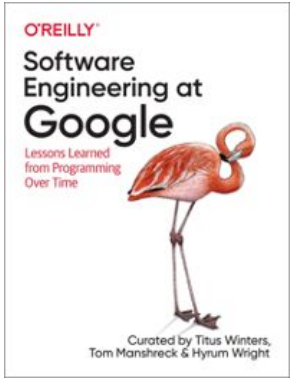
Companies using an RFC-like engineering planning process*

- Airbnb
- Affirm
- Algolia
- Amazon
- AutoScout24
- Asana
- Atlassian
- Blue Apron
- Bitrise
- Booking.com
- Brex
- BrowserStack
- Canonical
- Carousell
- Catawiki
- Cazoo
- Cisco
- CockroachDB
- Coinbase
- Comcast Cable
- Container Solutions
- Contentful
- Couchbase
- Criteo
- Curve
- Daimler
- Delivery Hero
- Doctolib
- DoorDash
- Dune Analytics
- eBay
- Ecosia
- Elastic
- Expedia
- Glovo
- Gojek
- Grab
- Faire
- Flexport
- GitHub
- GitLab
- GoodNotes
- Google
- Grafana Labs
- GrubHub
- HashiCorp
- Hopin
- Hudl
- Indeed
- Intercom
- LinkedIn
- Kiwi.com
- Klarna
- MasterCard
- Mews
- MongoDB
- Monzo
- Mollie
- Miro
- N26
- Netlify
- Nobl9
- Notion
- Nubank
- Oscar Health
- Octopus Deploy
- OLX
- Onfido
- Pave
- Peloton
- Picnic
- PlanGrid
- Preply
- Razorpay
- Reddit
- Red Hat
- SAP
- Salesforce
- Shopify
- Siemens
- Spotify
- Square
- Stripe
- Synopsys
- Skyscanner
- SoundCloud
- Sourcegraph
- Spotify
- Stedi
- Stream
- SumUp
- Thumbtack
- TomTom
- Trainline
- TrueBill
- Trustpilot
- Twitter
- Uber
- VanMoof
- Virta Health
- VMWare
- Wayfair
- Wave
- Wise
- WarnerMedia & HBO
- Zalando
- Zapier
- Zendesk
- Zillow

*not a complete list

pragmaticengineer.com

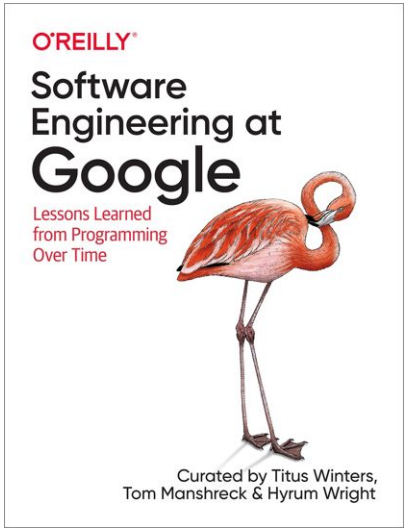
Design Documents



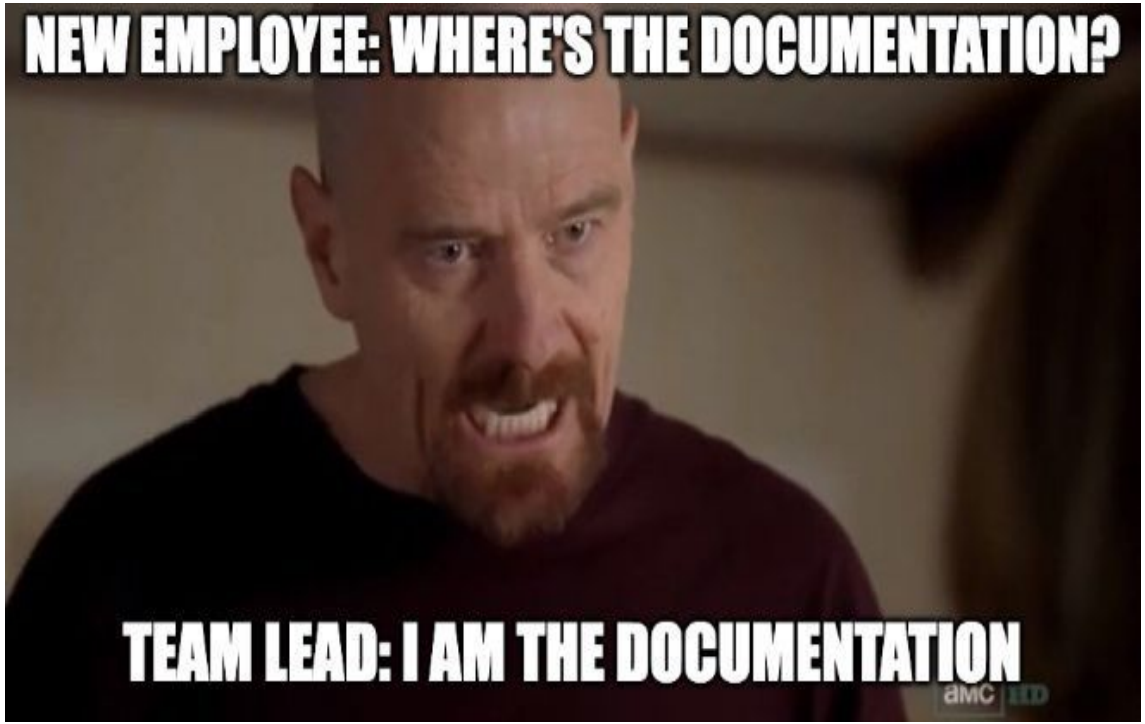
- Code review before there is code!
- Collaborative (Google Docs)
- Ensure various concerns are covered, such as: security implications, internationalization, storage requirements, and privacy concerns.
- A good design doc should cover
  - Goals and use cases for the design
  - Implementation ideas (not too specific!)
  - Propose key design decisions with an emphasis on their individual tradeoffs

Design Documents

- The *best* design docs suggest design goals, and cover alternative designs, documenting the strengths and weaknesses of each.
- The *worst* design docs accidentally embed ambiguities, which cause implementers to develop contradictory solutions that the customer doesn't want.



Why is this important?



Observe Sourcegraph Design Docs

- Docs are publicly available
  https://drive.google.com/drive/folders/1zP3FxdDIcSQGC1qvM9lHZRaHH4I9Jwwa

- Let’s take a look at one!

Common parts/templates

Metadata: *version, date, authors*

Executive Summary: *problem being solved, project mission*

Stakeholders (and non-stakeholders)

Scenarios

Non-Goals

Design Considerations and Tradeoffs

Open Issues

When to use an RFC:

Sourcegraph

- You want to frame a problem and propose a solution.
- You want thoughtful feedback from team members on our globally-distributed remote team.
- You want to surface an idea, tension, or feedback.
- You want to define a project or design brief to drive project collaboration.
- You need to surface and communicate around a highly cross-functional decision with our formal decision-making process.

Team Challenge: Document & Deploy

- Your challenge: Deploy a simple web app using a cloud provider of your choice
- Each team will:
    - Write a short RFC (*Design Doc*) capturing:
        - Why you chose the provider
            - Some options to consider: AWS, Azure, Google Cloud, Render, Vercel, Github codespaces
        - Trade-offs
        - Challenges and how they were resolved
        - Document the steps needed to deploy the app

- This is the web app: https://github.com/EduardoFF/albumy.git
- Create a **Google Doc** to serve as your team's **RFC** (Request for Comments).
- Collaborate in real time as you tackle the challenge. Document your decisions, steps taken, trade-offs, and challenges.
- Use comments to discuss alternatives, raise questions, or flag issues during the process.