CI and Cloud Computing

17-313 Fall 2025

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

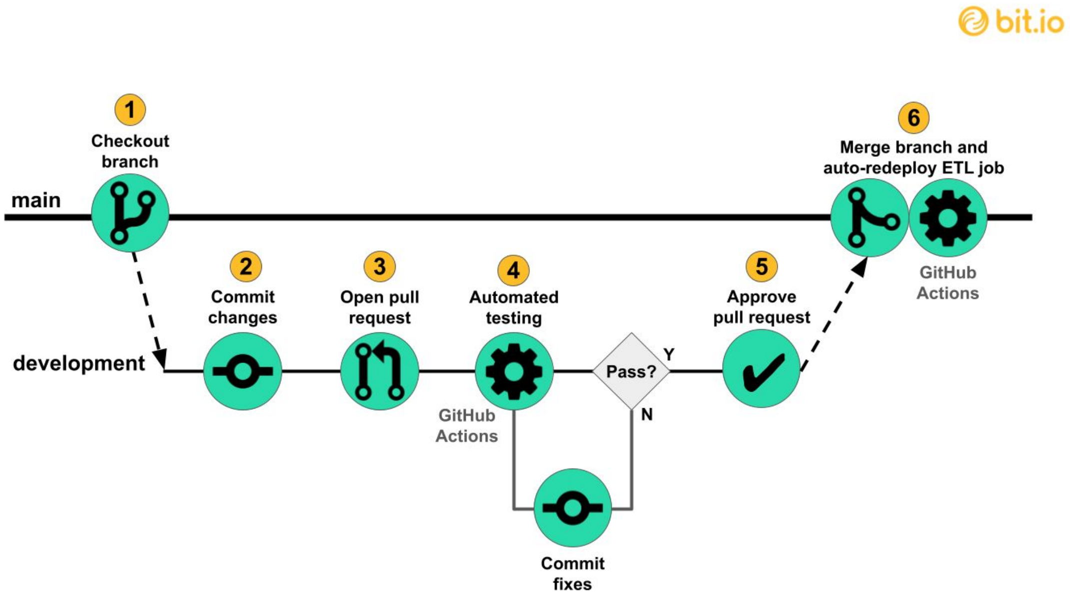
Continuous Integration:

Catch mistakes before you push your code!

History of CI

- (1999) Extreme Programming (XP) rule: “Integrate Often”
- 
  (2000) Martin Fowler posts “Continuous Integration” blog
- (2001) First CI tool
- (2005) Hudson/Jenkins
- (2011) Travis CI
- (2019) GitHub Actions

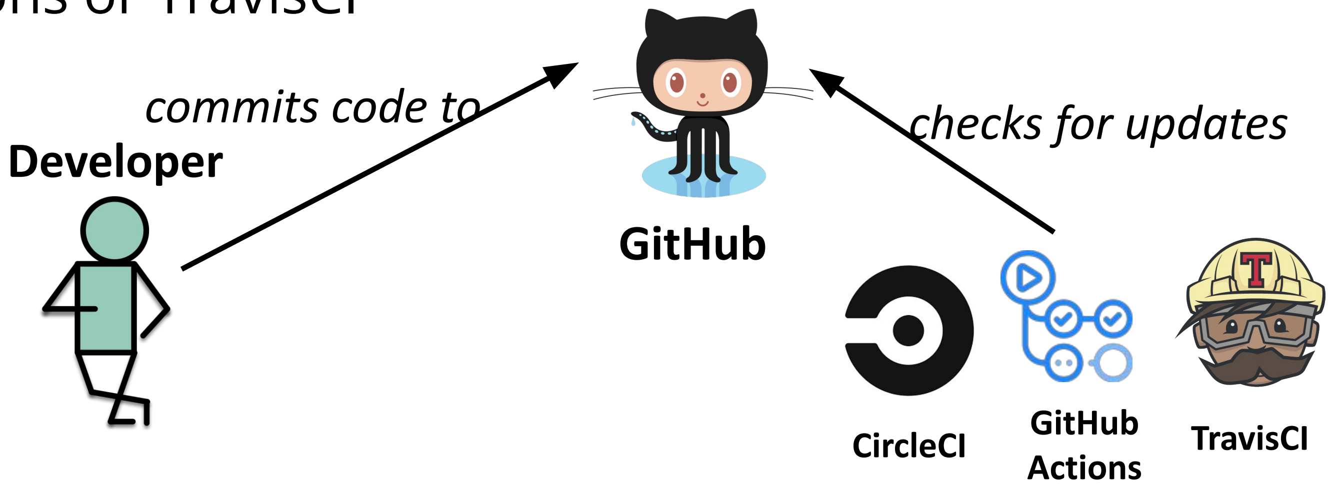
Example CI Workflow



Source: https://innerjoin.bit.io/making-a-simple-data-pipeline-part-4-ci-cd-with-github-actions-733251f211a6

CI is triggered by commits, pull requests, and other actions

Example: Small scale CI, with a service like CircleCI, GitHub Actions or TravisCI



Runs build for each commit

CI Research

Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility

Michael Hilton
Oregon State University, USA
mhilton@cmu.edu

Nicholas Nelson
Oregon State University, USA
nelsonni@oregonstate.edu

Timothy Tunnell
University of Illinois, USA
tunnell2@illinois.edu

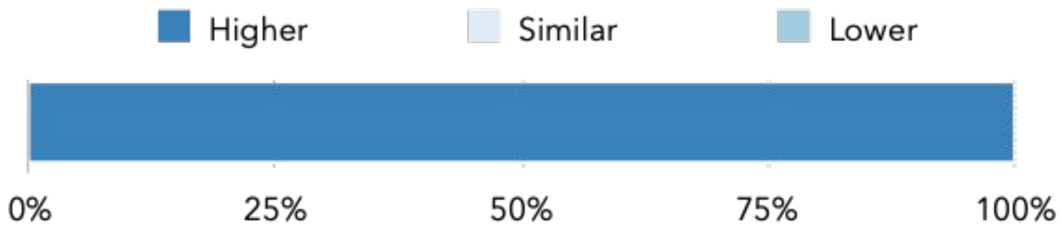
Darko Marinov
University of Illinois, USA
marinov@illinois.edu

Danny Dig
Oregon State University, USA
digd@oregonstate.edu

"523 complete responses, and a total of 691 survey responses from over 30 countries. Over 50% of our participants had over 10 years of software development experience, and over 80% had over 4 years of experience."

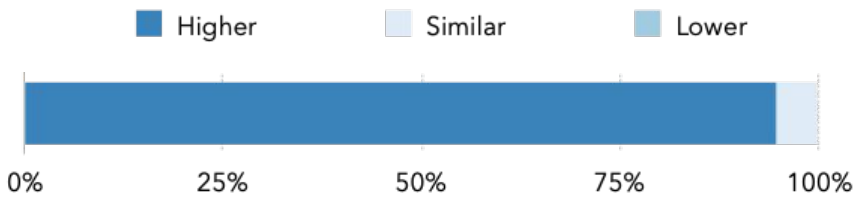
Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?



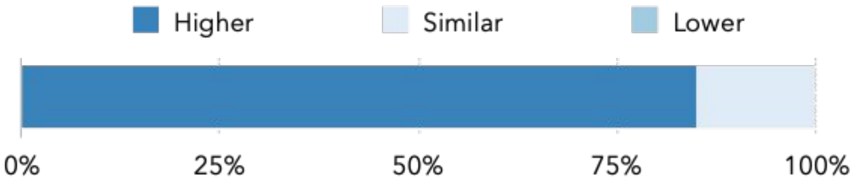
Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?
Do projects with CI have (higher/similar/lower) test quality?



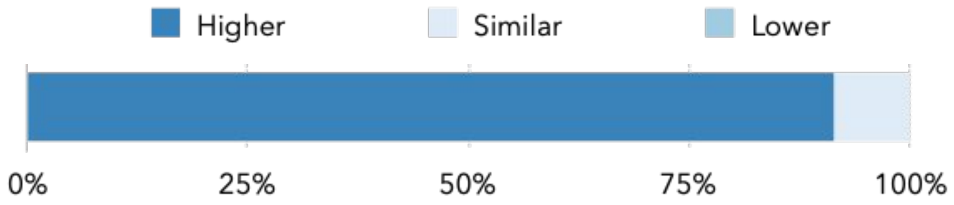
Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?
Do projects with CI have (higher/similar/lower) test quality?
Do projects with CI have (higher/similar/lower) code quality?



Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?
Do projects with CI have (higher/similar/lower) test quality?
Do projects with CI have (higher/similar/lower) code quality?
Are developers on projects with CI (more/similar/less) productive?



Agile values fast quality feedback loops

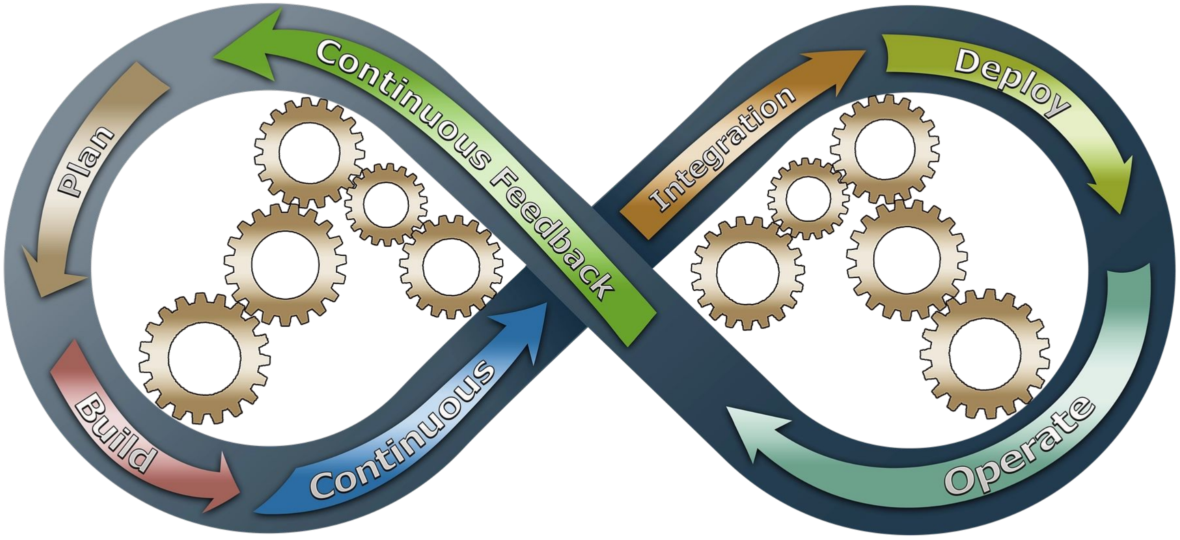
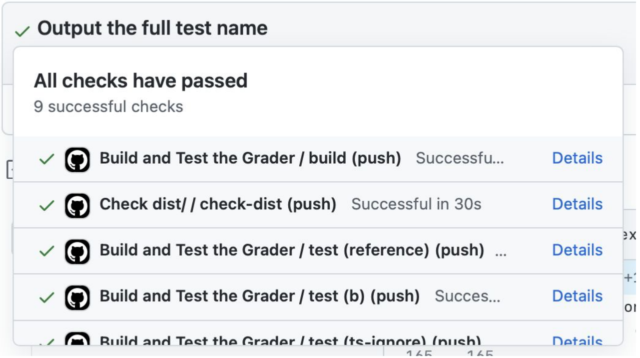
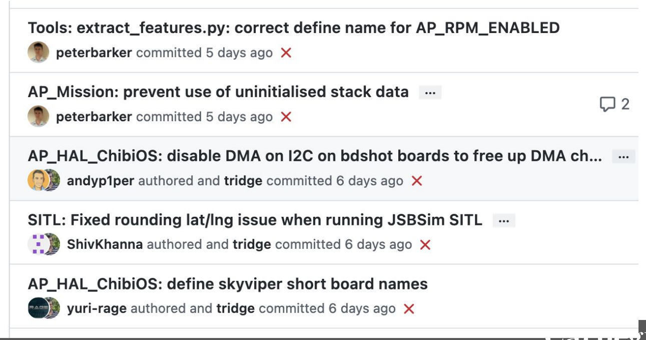


Image source: https://sdtimes.com/devops/feedback-loops-are-a-prerequisite-for-continuous-improvement/

Attributes of effective CI processes

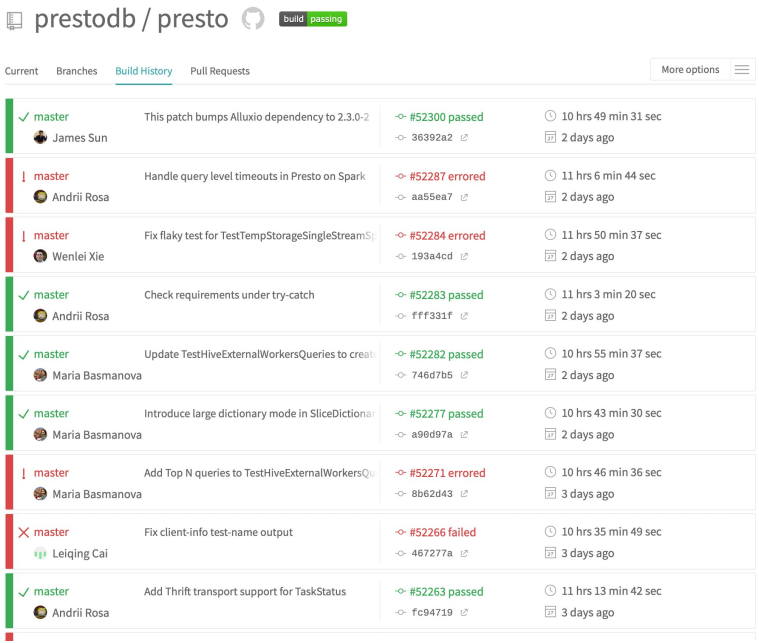
- Policies:
  - **Do not** allow builds to **<u>remain broken</u>** for a long time
  - CI should run for every change
  - CI should not completely replace pre-commit testing
- Infrastructure:
  - CI should be fast, providing feedback within minutes or hours
  - CI should be repeatable (**deterministic**)





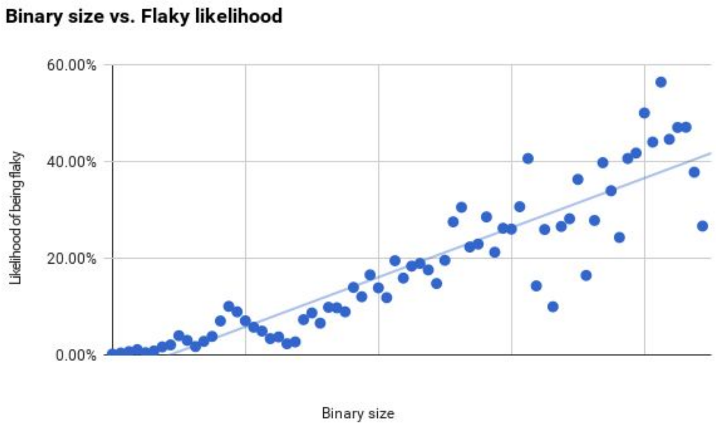
Effective CI processes are run often enough to reduce debugging effort

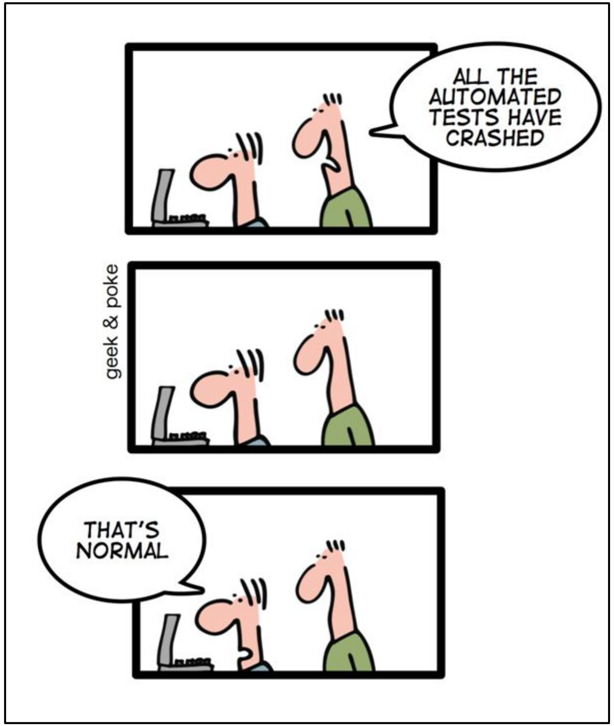
- Failed CI runs indicate a bug was introduced, and **caught in that run**
- **More changes per-CI run** require **more manual debugging** effort to assign blame
- A single change per-CI run pinpoints the culprit



Challenge: Flaky Tests

"Google has around 4.2 million tests that run on our continuous integration system. Of these, around 63 thousand have a flaky run over the course of a week"

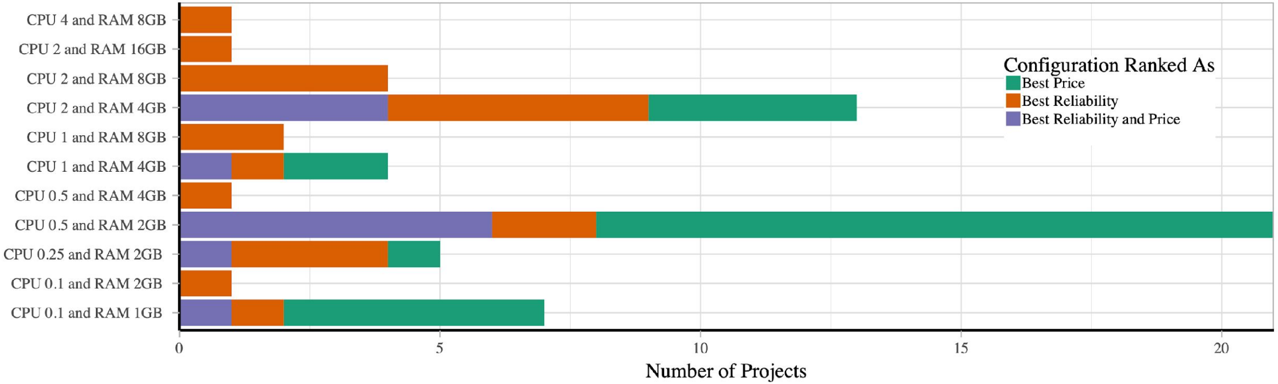




https://testing.googleblog.com/2017/04/where-do-our-flaky-tests-come-from.html

Effective CI processes allocate enough resources to mitigate flaky tests

- *Flaky* tests might be dependent on timing (failing due to timeouts)
- Running tests without enough CPU/RAM can result in increased flaky failure rates and unreliable builds

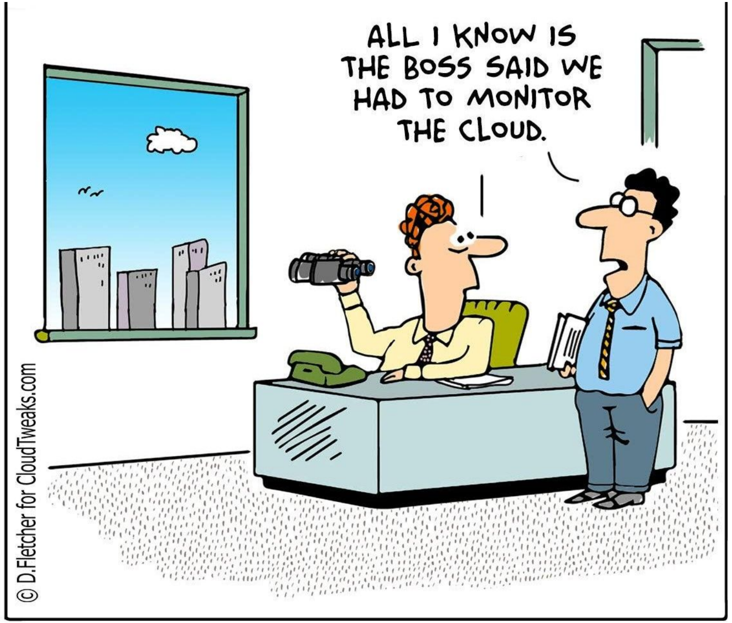


"The Effects of Computational Resources on Flaky Tests", Silva et al

Cloud Computing enables Continuous Integration and Deployment/Delivery

Cloud Computing

in a Nutshell



1970s Teleprocessing

Photo Credit: University of

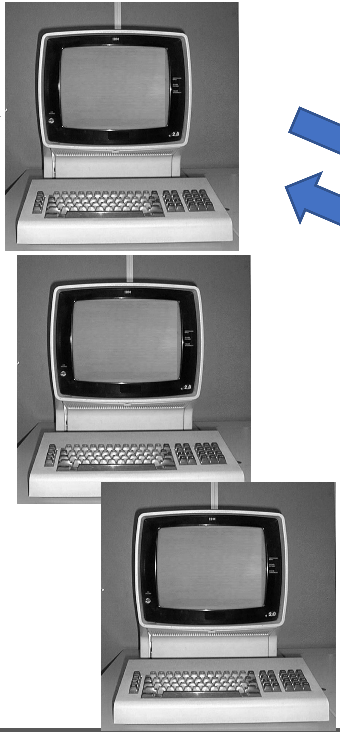


Photo Credit: ArnoldReinhold, CC BY-SA 3.0 via Wikimedia Commons

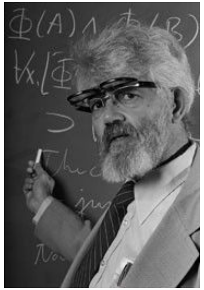
Photo Credit: Wikipedia

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

2000s Cloud Computing

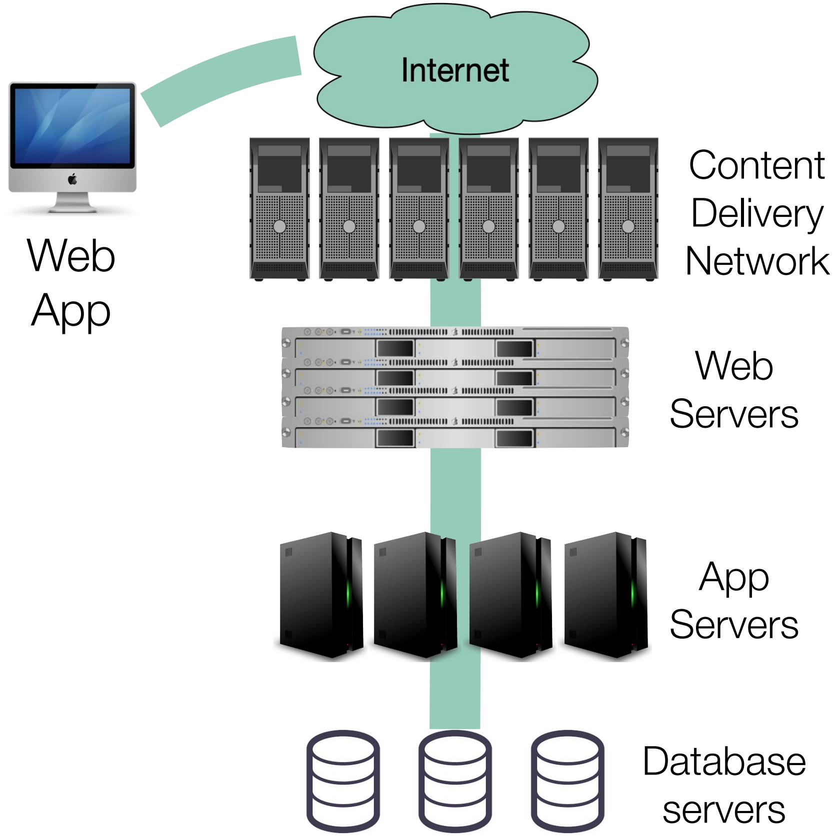


"Computing may someday be organized as a public utility just as the telephone system is a public utility...Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system ..."

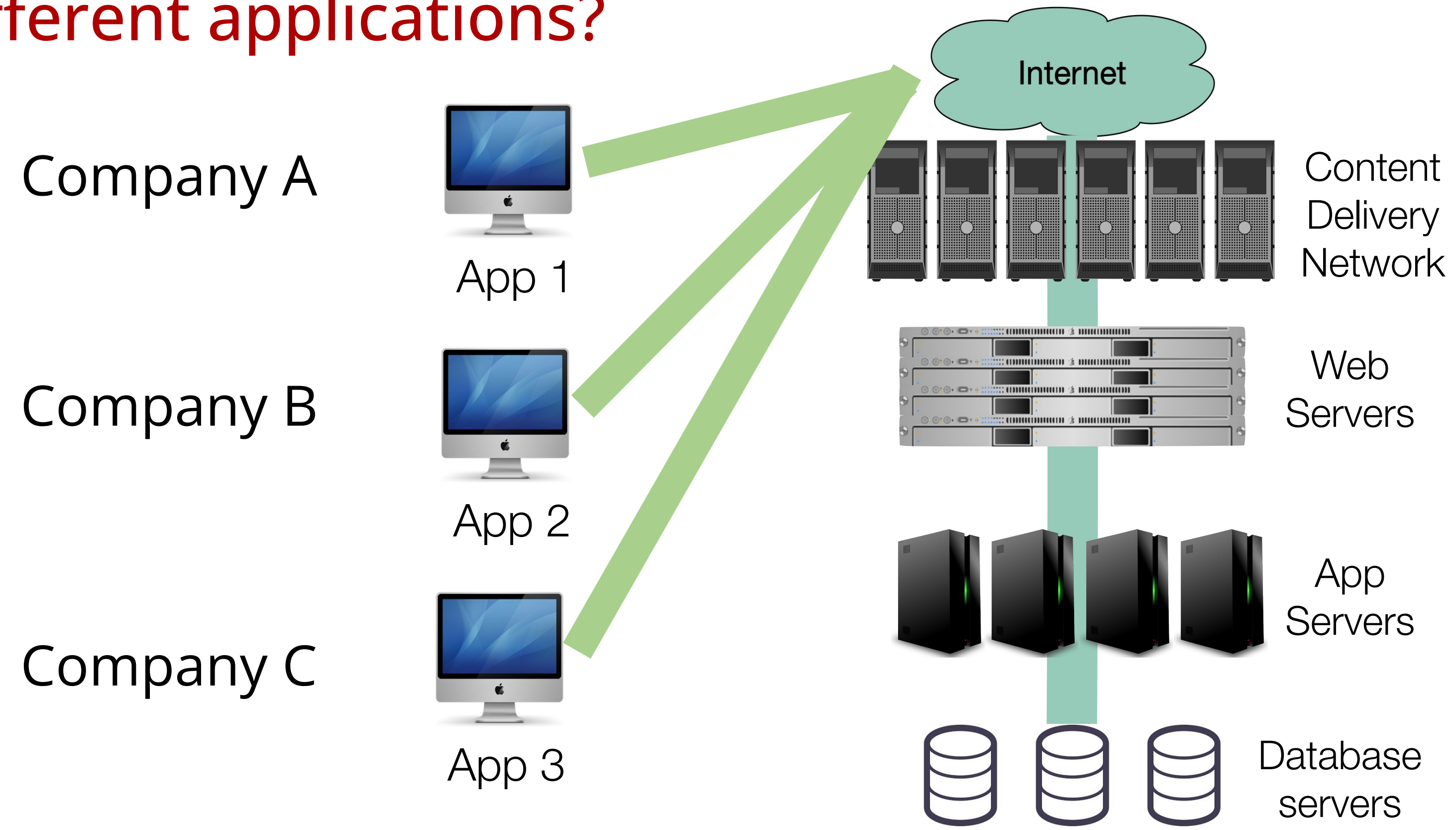
McCarthy's predictions come true!

A traditional deployment of a Web Application

- 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 parts of this infrastructure can be shared across different applications?



Multi-Tenancy creates economies of scale

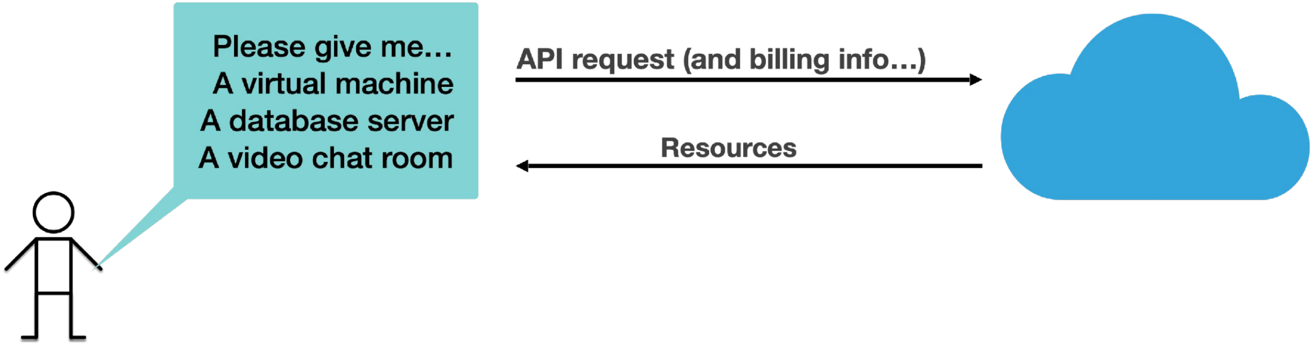
- At the physical level:
  - Multiple customers’ **physical machines** in the same data center
  - Save on physical costs (centralize power, cooling, security, maintenance)
- At the physical server level:
  - Multiple customers’ **virtual machines** in the same physical machine
  - Save on resource costs (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)

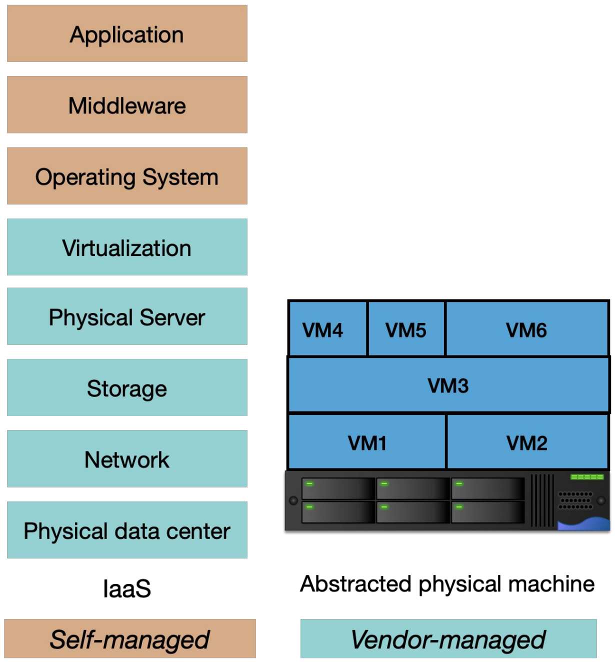
Cloud services gives on-demand access to infrastructure, “as a service”

- Vendor provides a service catalog of “*X as a service*” abstractions that provide infrastructure as a service
- API allows 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



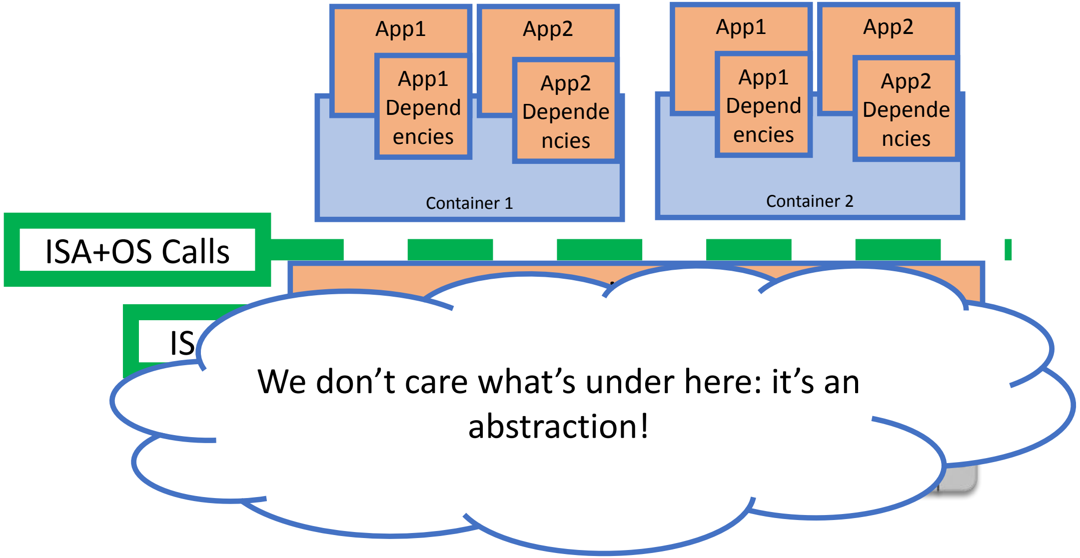
Virtual Machines to Containers

- 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)

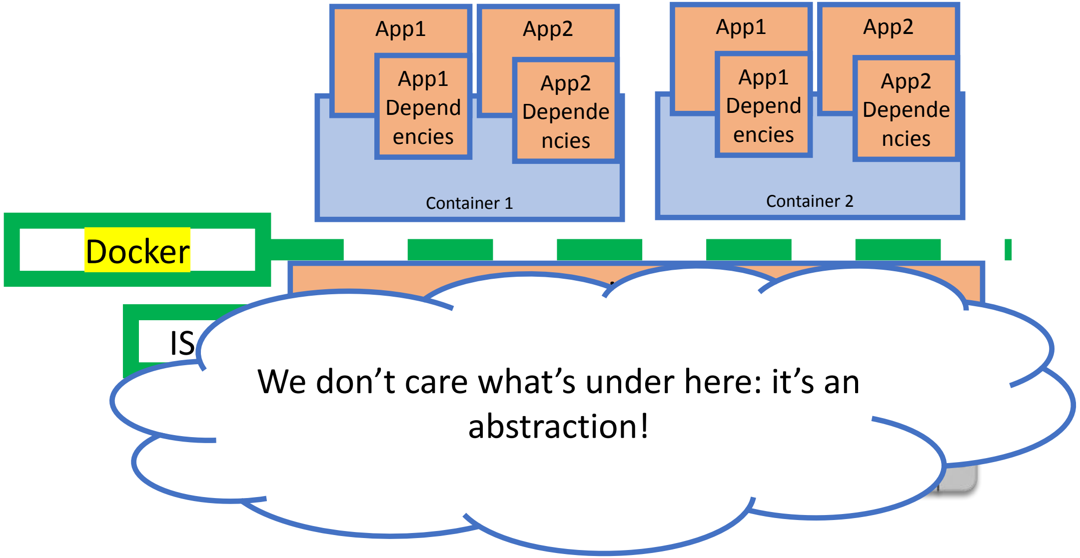
CaaS: Containers as a Service

- 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")



A container contains your apps and all their dependencies

- Each application is encapsulated in a “lightweight container,” includes:
  - System libraries (e.g. glibc)
  - External dependencies (e.g. nodejs)
- “Lightweight” in that container images are smaller than VM images - multi tenant containers run in the OS
- Cloud providers offer “containers as a service” (Amazon ECS Fargate, Azure Kubernetes, Google Kubernetes)

NodeBB / Dockerfile

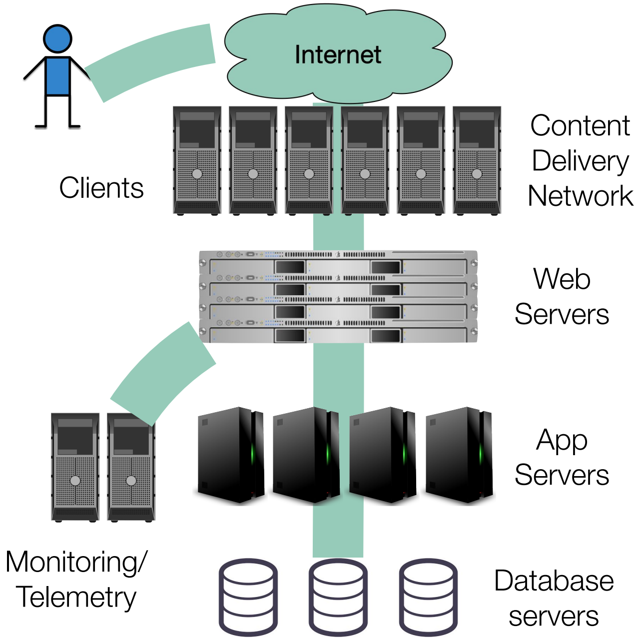
angelaz1 Initial NodeBB Commit b6951a8 · last year History

Code Blame 25 lines (16 loc) · 485 Bytes Raw

```dockerfile
FROM node:lts

RUN mkdir -p /usr/src/app && \
    chown -R node:node /usr/src/app
WORKDIR /usr/src/app

ARG NODE_ENV
ENV NODE_ENV $NODE_ENV

COPY --chown=node:node install/package.json /usr/src/app/package.json

USER node

RUN npm install --only=prod && \
    npm cache clean --force

COPY --chown=node:node . /usr/src/app

ENV NODE_ENV=production \
    daemon=false \
    silent=false

EXPOSE 4567

CMD test -n "${SETUP}" && ./nodebb setup || node ./nodebb build; node ./nodebb start
```

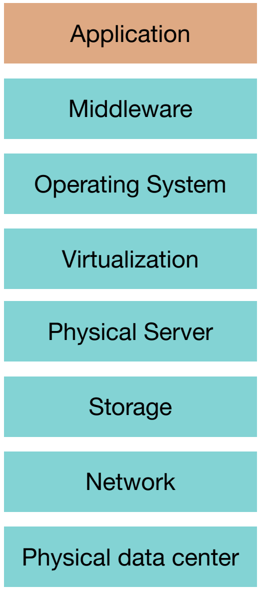
Platform-as-a-Service: vendor supplies OS + middleware

- 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



PaaS

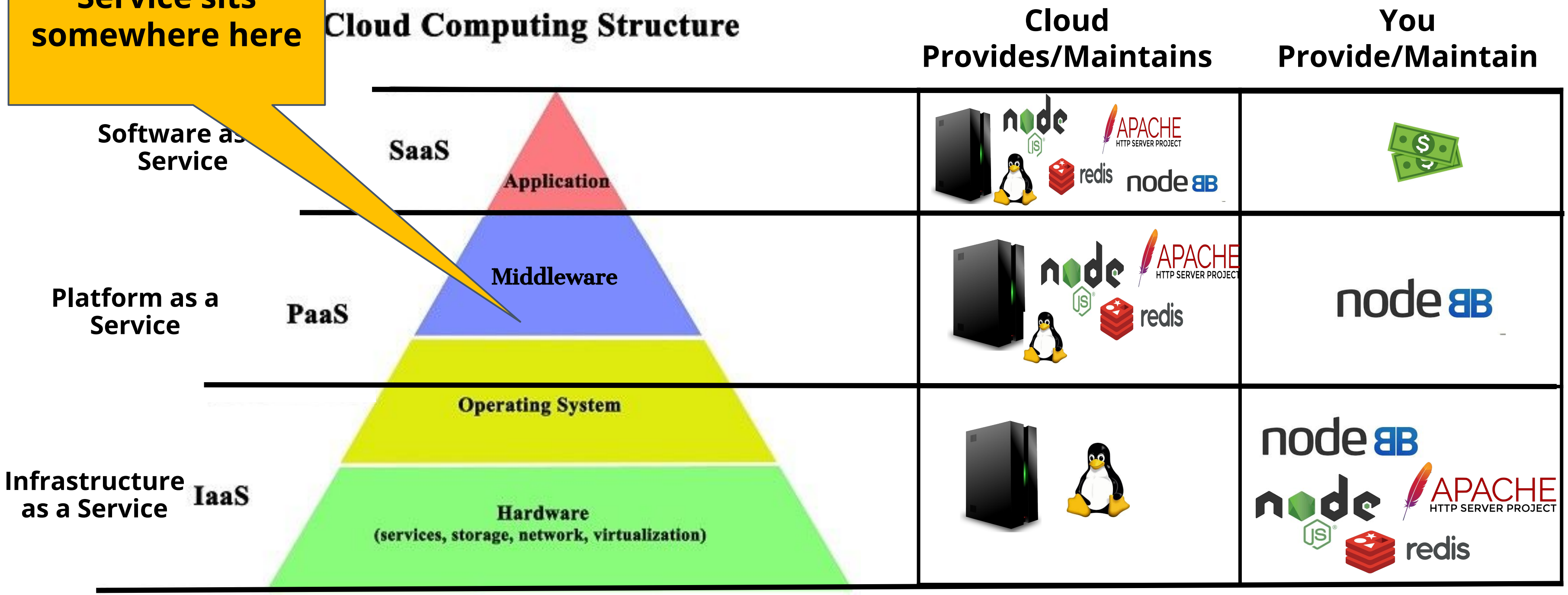
Software as a Service

- Software that is fully built, deployed, and maintained by a provider, and offered directly to end-users over the internet (e.g., Gmail, Google Docs, Slack, Zoom)
- User Perspective:
  - Access through a browser or app.
  - No need to install, update, or manage servers.
  - Pay as you go (subscription model).

The interesting engineering work happens at lower layers (IaaS or PaaS), where you build and deploy software systems.

Cloud Computing: Analogy using NodeBB

Container as a Service sits somewhere here



Cloud Infrastructure is best for variable workloads

- Consider:
  - Does your workload benefit from ability to scale up or down?
  - Variable workloads have different demands over time (most common)
  - Constant workloads require sustained resources (less common)
- Example:
  - Need to run 300 VMs, each 4 vCPUs, 16GB RAM
- Private cloud:
  - Dell PowerEdge Pricing (AMD EPYC 64 core CPUs)
  - 7 servers, each 128 cores, 512GB RAM, 3 TB storage = $162,104
- Public cloud:
  - Amazon EC2 Pricing (M7a.xlarge instances, $0.153/VM-hour)
  - 10 VMs for 1 year + 290 VMs for 1 month: $45,792.90
  - 300 VMs for 1 year: $402,084.00

Public clouds are not the only option

- “Public” clouds are connected to the internet and available for anyone to use
  - Examples: Amazon, Azure, Google Cloud, DigitalOcean
- “Private” clouds use cloud technologies with on-premises, self-managed hardware
  - Cost-effective when a large scale of baseline resources are needed
  - Example management software: OpenStack, VMWare, Proxmox, Kubernetes
- “Hybrid” clouds integrate private and public (or multiple public) clouds
  - Effective approach to “burst” capacity from private cloud to public cloud