

# 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!

5

# History of CI



(1999) Extreme Programming (XP) rule: “Integrate Often”



(2000) Martin Fowler posts “[Continuous Integration](#)” blog



(2001) First CI tool



**Jenkins**

(2005) Hudson/Jenkins



**Travis CI**

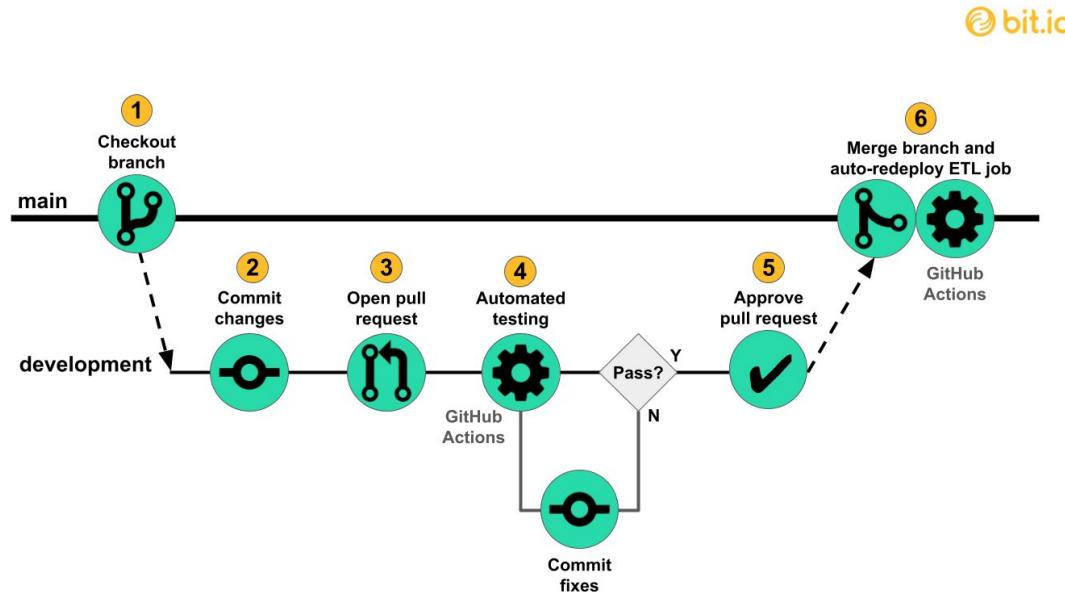
(2011) Travis CI



GitHub Actions

(2019) GitHub Actions

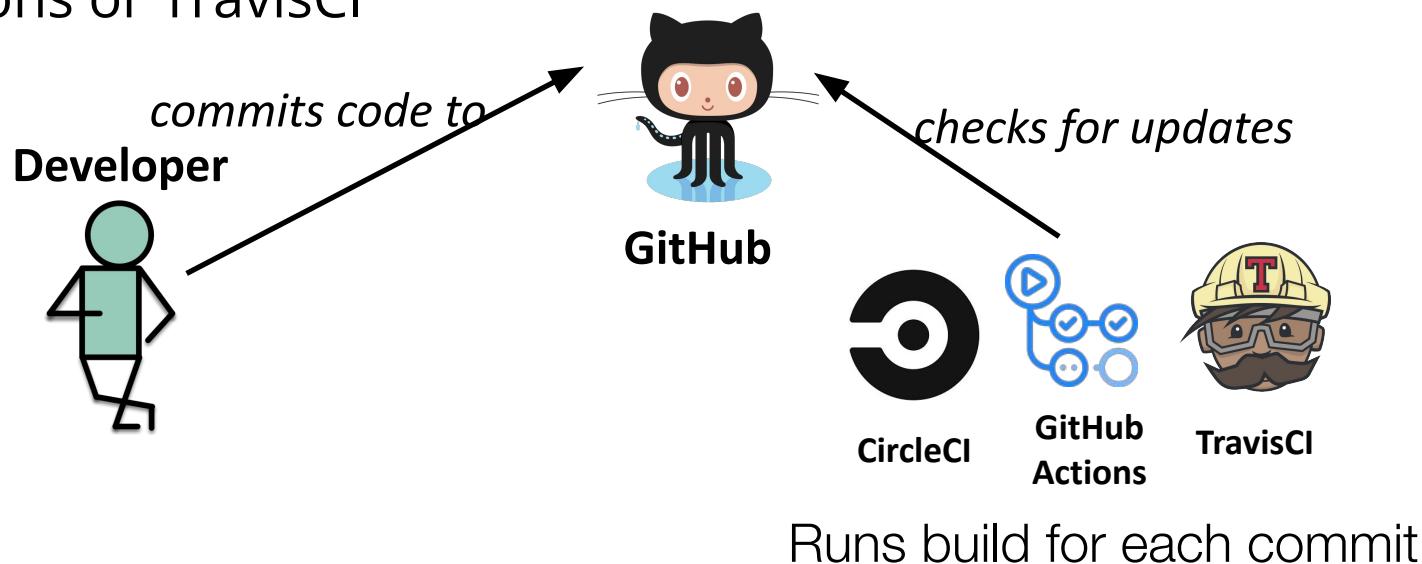
# Example CI Workflow



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



# CI Research

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

Michael Hilton  
Oregon State University, USA  
[mhilton@cmu.edu](mailto:mhilton@cmu.edu)

Nicholas Nelson  
Oregon State University, USA  
[nelsonni@oregonstate.edu](mailto:nelsonni@oregonstate.edu)

Timothy Tunnell  
University of Illinois, USA  
[tunnell2@illinois.edu](mailto:tunnell2@illinois.edu)

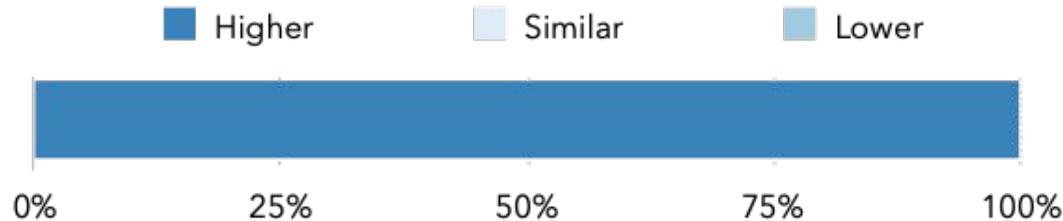
Darko Marinov  
University of Illinois, USA  
[marinov@illinois.edu](mailto:marinov@illinois.edu)

Danny Dig  
Oregon State University, USA  
[digd@oregonstate.edu](mailto: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.”*

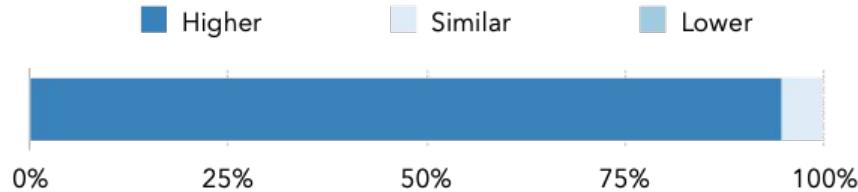
# Developers report:

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



# Developers report:

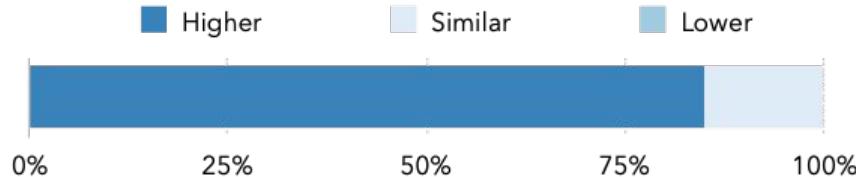
Do developers on projects with CI give (more/similar/less) value to automated tests?  
Do projects with CI have (higher/similar/lower) test quality?



# 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?**



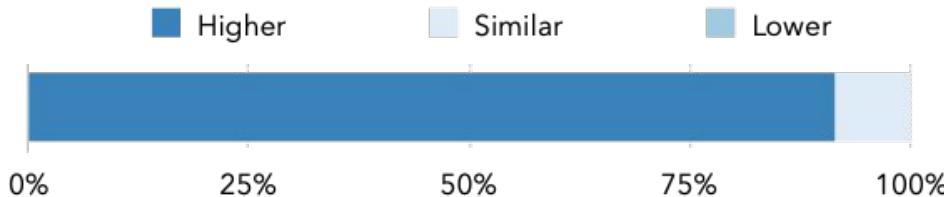
# Developers report:

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

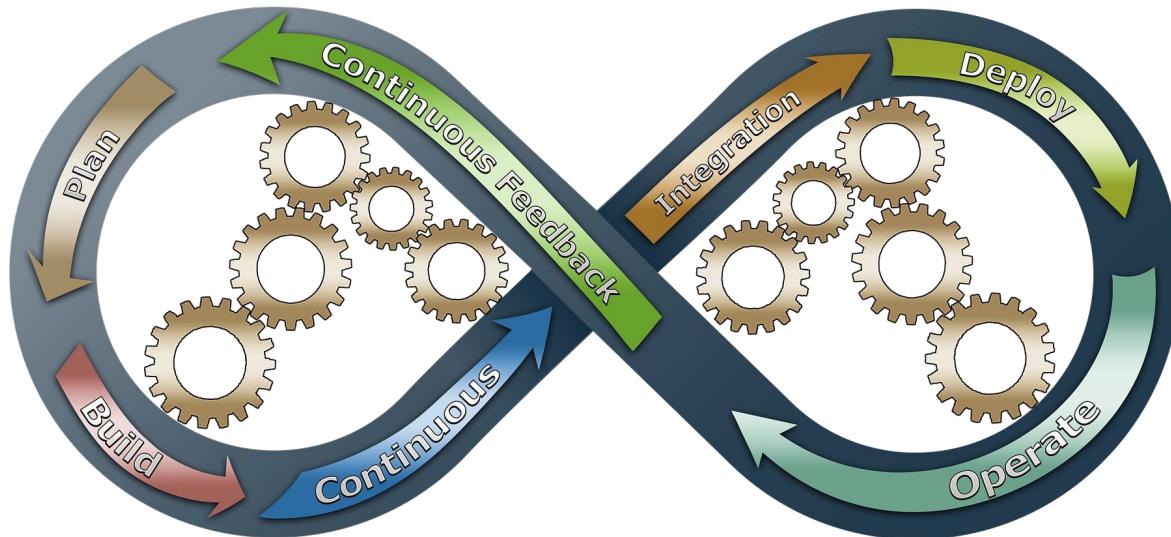


Image source: <https://sdtimes.com/devops/feedback-loops-are-a-prerequisite-for-continuous-improvement/>

# Attributes of effective CI processes

- Policies:
  - **Do not allow builds to remain broken 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**)

A screenshot of a CI status page. At the top, a green checkmark icon and the text "Output the full test name" are displayed. Below this, a summary states "All checks have passed" and "9 successful checks". A list of five build steps is shown, each with a green checkmark, a small circular icon, and the step name followed by "(push)". To the right of each step name are "Successful" or "Successf..." and "Details" links. The steps are: "Build and Test the Grader / build (push)", "Check dist / check-dist (push)", "Build and Test the Grader / test (reference) (push)", "Build and Test the Grader / test (b) (push)", and "Build and Test the Grader / test (ts-ignore) (push)".

A screenshot of a GitHub pull request interface. At the top, there is a commit message: "Tools: extract\_features.py: correct define name for AP\_RPM\_ENABLED". It shows a profile picture for "peterbarker", the text "committed 5 days ago", and a red 'X' icon. Below this, another commit message is shown: "AP\_Mission: prevent use of uninitialised stack data". It also shows a profile picture for "peterbarker", the text "committed 5 days ago", and a red 'X' icon. There are two small blue speech bubble icons with the number "2" next to them. Further down, another commit message is shown: "AP\_HAL\_ChibiOS: disable DMA on I2C on bdshot boards to free up DMA ch...". It shows a profile picture for "andy1per", the text "authored and tridge committed 6 days ago", and a red 'X' icon. Below this, another commit message is shown: "SITL: Fixed rounding lat/lng issue when running JSBSim SITL". It shows a profile picture for "ShivKhanna", the text "authored and tridge committed 6 days ago", and a red 'X' icon. Finally, another commit message is shown: "AP\_HAL\_ChibiOS: define skyviper short board names". It shows a profile picture for "yuri-rage", the text "authored and tridge committed 6 days ago", and a red 'X' icon.

# Effective CI processes are run often enough to reduce debugging effort

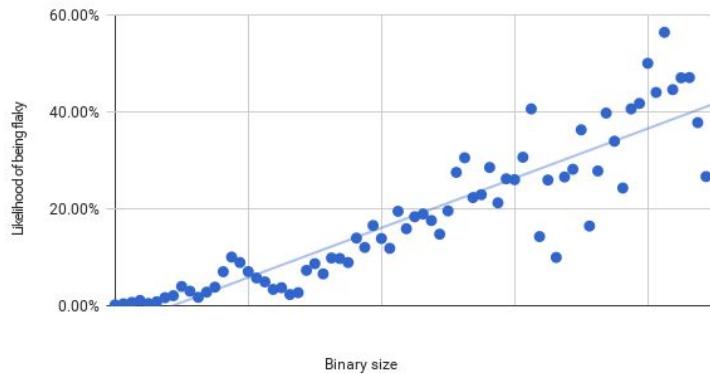
- Failed CI runs indicate a bug was introduced, and **caught in that run**
- **More changes per-CI run require more manual debugging** effort to assign blame
- A single change per-CI run pinpoints the culprit

prestodb / presto			
	Current Branches Build History Pull Requests	More options	Build Status
✓ master	This patch bumps Alluxio dependency to 2.3.0-2 James Sun	#52300 passed 36392a2	10 hrs 49 min 31 sec 2 days ago
✗ master	Handle query level timeouts in Presto on Spark Andrii Rosa	#52287 errored aa55ea7	11 hrs 6 min 44 sec 2 days ago
✗ master	Fix flaky test for TestTempStorageSingleStreamS... Wenlei Xie	#52284 errored 193a4cd	11 hrs 50 min 37 sec 2 days ago
✓ master	Check requirements under try-catch Andrii Rosa	#52283 passed ffff331f	11 hrs 3 min 20 sec 2 days ago
✓ master	Update TestHiveExternalWorkersQueries to creat... Maria Basanova	#52282 passed 746d7b5	10 hrs 55 min 37 sec 2 days ago
✓ master	Introduce large dictionary mode in SliceDictionary Maria Basanova	#52277 passed a90d97a	10 hrs 43 min 30 sec 2 days ago
✗ master	Add Top N queries to TestHiveExternalWorkersQu... Maria Basanova	#52271 errored Bb82d43	10 hrs 46 min 36 sec 3 days ago
✗ master	Fix client-info test-name output Lelqing Cai	#52266 failed 467277a	10 hrs 35 min 49 sec 3 days ago
✓ master	Add Thrift transport support for TaskStatus Andrii Rosa	#52263 passed fc94719	11 hrs 13 min 42 sec 3 days ago

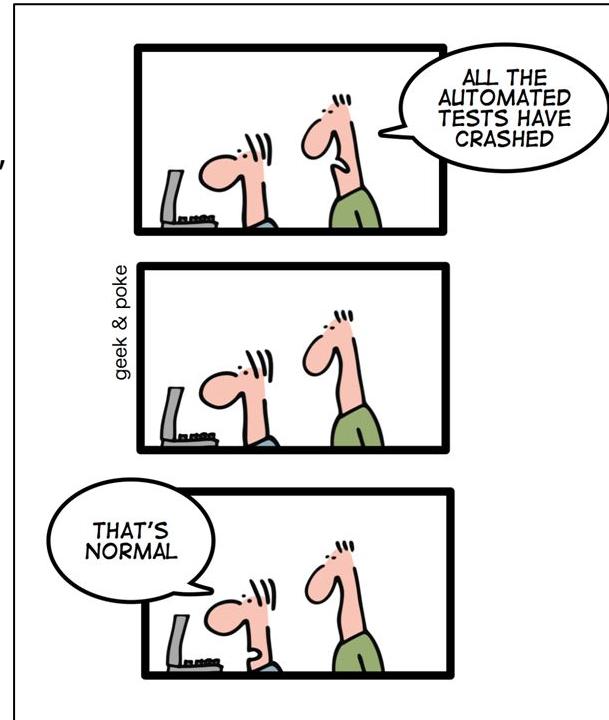
# Challenge: Flaky Tests

"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"

Binary size vs. Flaky likelihood

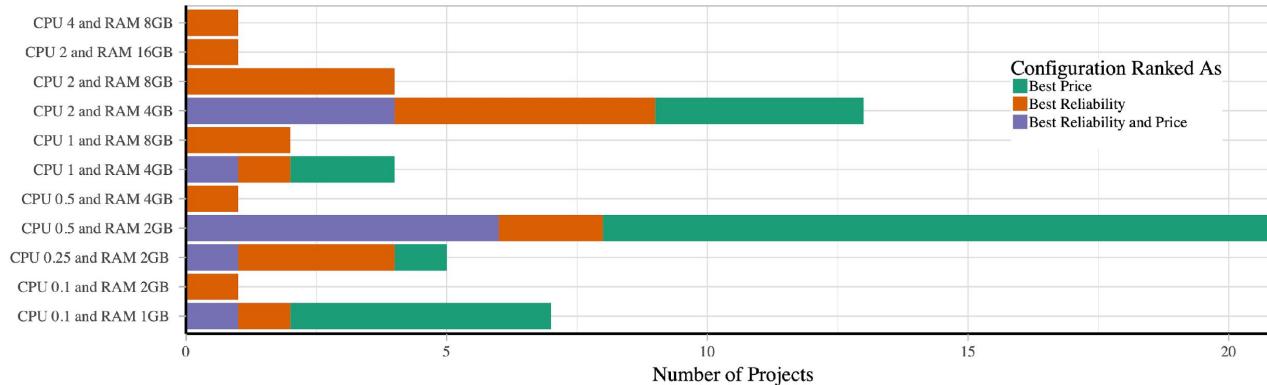


<https://testing.googleblog.com/2017/04/where-do-our-flaky-tests-come-from.html>



# 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



# Cloud Computing enables Continuous Integration and Deployment/Delivery

# Cloud Computing

in a Nutshell



# 1970s Teleprocessing

Photo Credit: University of

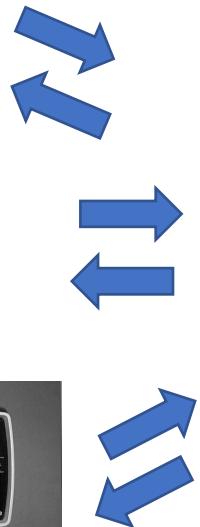


Photo Credit: ArnoldReinhold, CC BY-SA 3.0 via Wikimedia Commons



Photo Credit:  
[Wikipedia](#)

# 1980s & 1990s Personal Computing

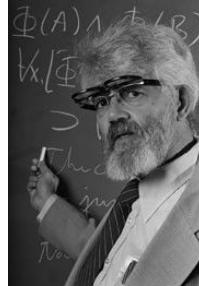
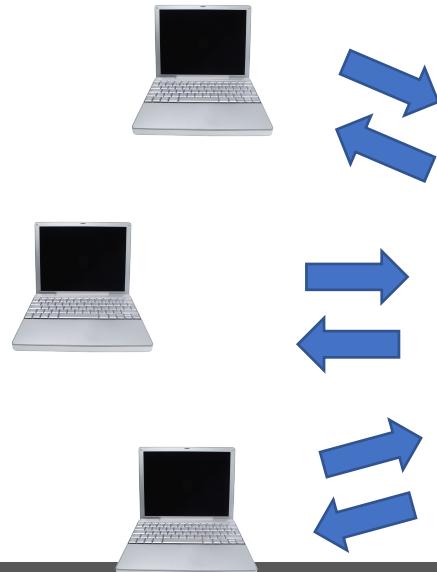


Photo Credit: Rama & Musée Bolo, [CC BY-SA 2.0 FR](#) via Wikimedia Commons



Photo Credit: Alexander Schaelss, [CC BY-SA 3.0](#) via Wikimedia Commons

# 2000s Cloud Computing

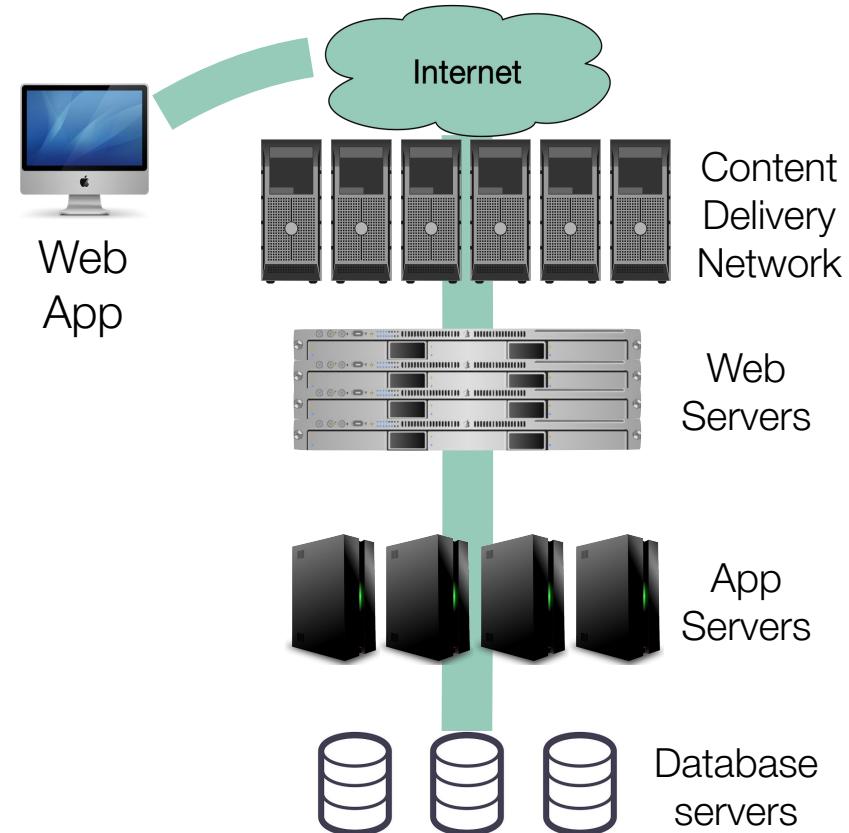


*"Computing may someday be organized as a public utility just as the telephone system is a public utility...Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system ..."*

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?

Company A



App 1

Company B

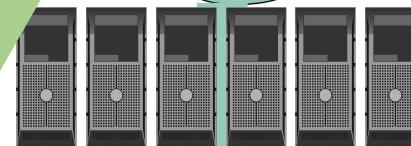


App 2

Company C



App 3



Content  
Delivery  
Network



Web  
Servers



App  
Servers



Database  
servers

# Multi-Tenancy creates economies of scale

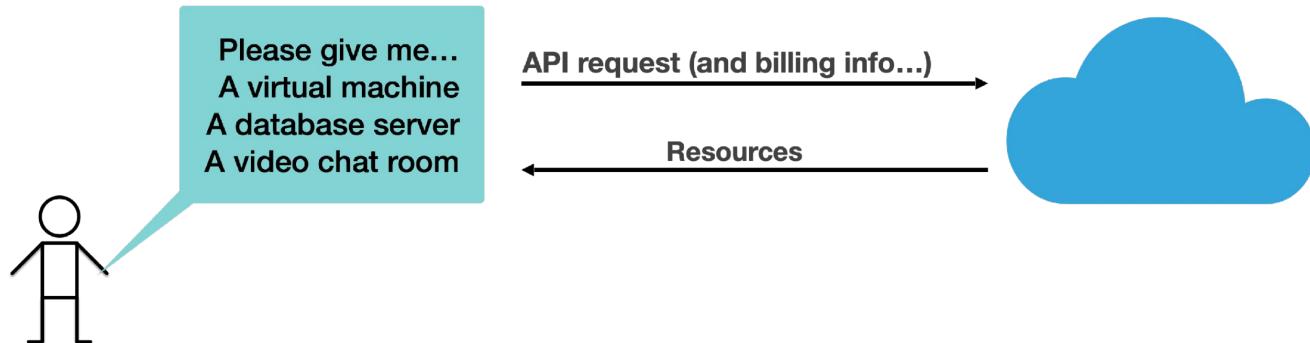
- At the physical level:
  - Multiple customers' **physical machines** in the same data center
  - Save on physical costs (centralize power, cooling, security, maintenance)
- At the physical server level:
  - Multiple customers' **virtual machines** in the same physical machine
  - Save on resource costs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

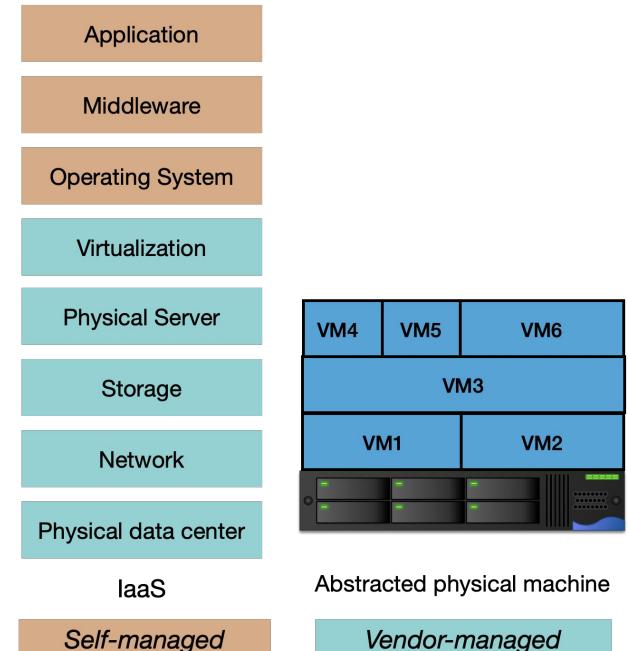
# Cloud services gives on-demand access to infrastructure, “as a service”

- 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

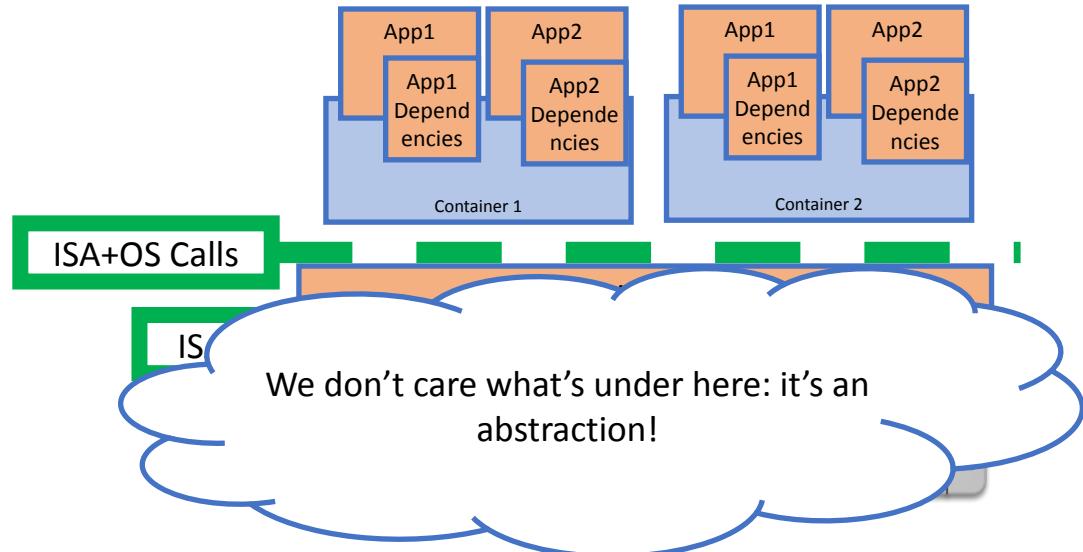


# Virtual Machines to Containers

- 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)

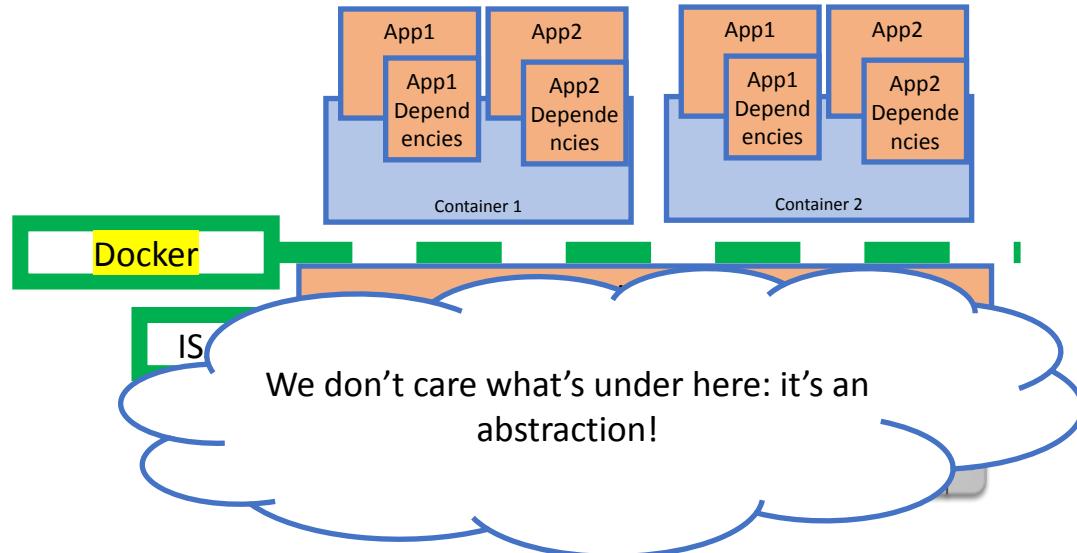
# CaaS: Containers as a Service

- 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)



angelaz1 Initial NodeBB Commit

b6951a8 · last year ⏲ History

Code

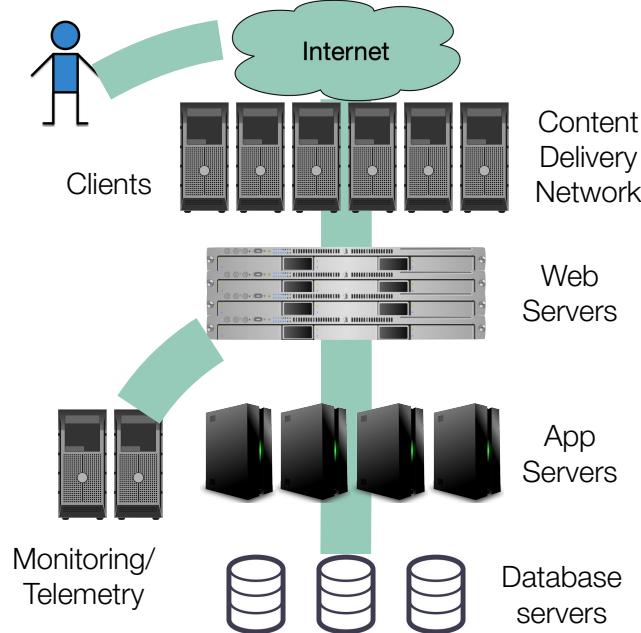
Blame 25 lines (16 loc) · 485 Bytes

Raw ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

```
1  FROM node:lts
2
3  RUN mkdir -p /usr/src/app && \
4      chown -R node:node /usr/src/app
5  WORKDIR /usr/src/app
6
7  ARG NODE_ENV
8  ENV NODE_ENV $NODE_ENV
9
10 COPY --chown=node:node install/package.json /usr/src/app/package.json
11
12 USER node
13
14 RUN npm install --only=prod && \
15     npm cache clean --force
16
17 COPY --chown=node:node . /usr/src/app
18
19 ENV NODE_ENV=production \
20     daemon=false \
21     silent=false
22
23 EXPOSE 4567
24
25 CMD test -n "${SETUP}" && ./nodebb setup || node ./nodebb build; node ./nodebb start
```

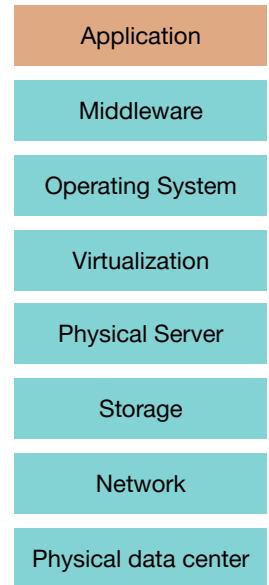
# Platform-as-a-Service: vendor supplies OS + middleware

- 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 PaaS 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 Computing Structure		Cloud Provides/Maintains	You Provide/Maintain
Software as a Service	SaaS Application	     nodeBB	
Platform as a Service	PaaS Middleware	    	nodeBB
Infrastructure as a Service	IaaS Operating System		nodeBB
	Hardware (services, storage, network, virtualization)		  

# 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