

CI and Cloud Computing

17-313 Fall 2024

Foundations of Software Engineering

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

Eduardo Feo Flushing

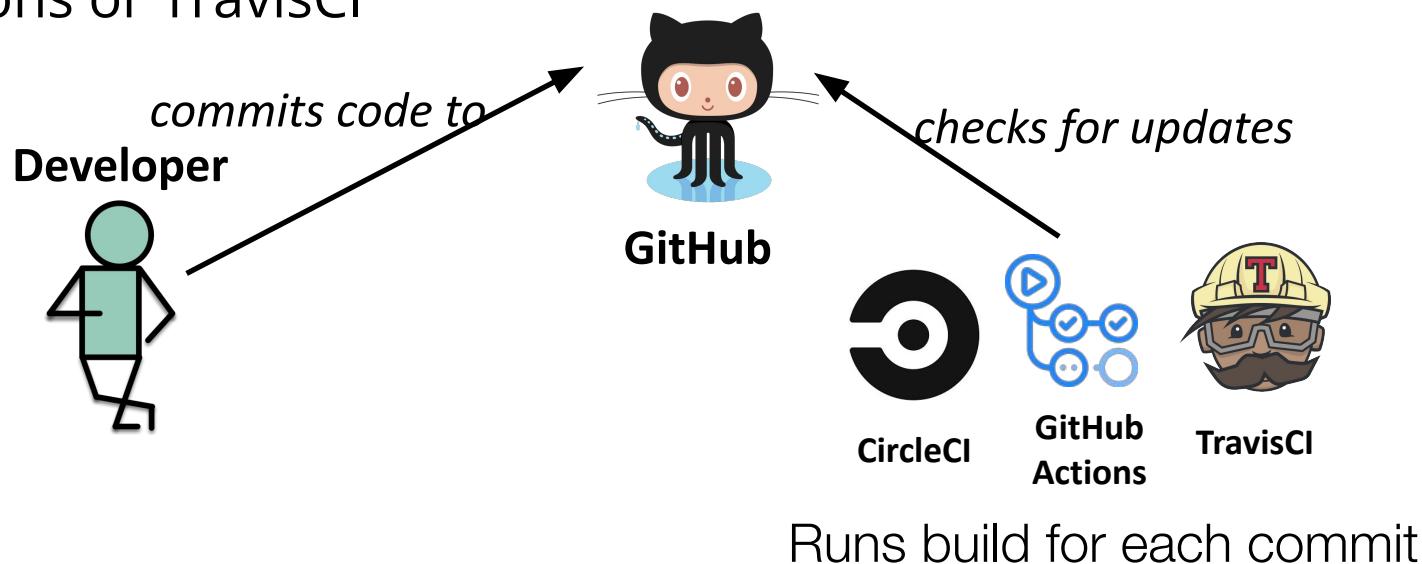
Review: Continuous Integration

Observation

**CI helps us catch errors
before others see them**

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

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



Agile values fast quality feedback loops

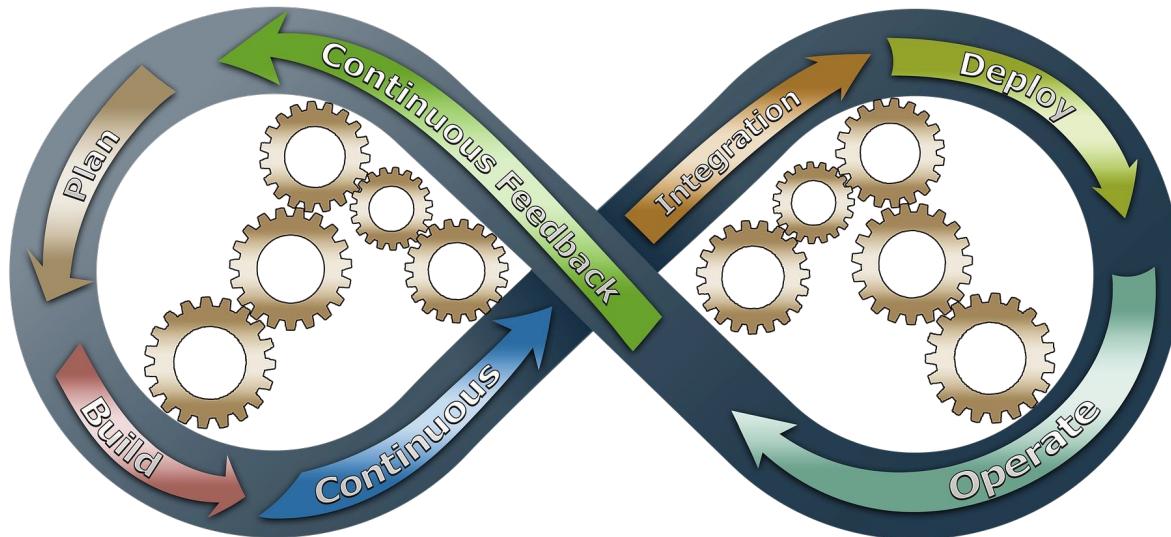


Image source: <https://sdtimes.com/devops/feedback-loops-are-a-prerequisite-for-continuous-improvement/>

Automating Feedback Loops is Powerful

Consider tasks that are done by *dozens* of developers (e.g. testing/deployment)

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS	
	1 HOUR	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS	
	6 HOURS			2 MONTHS	2 WEEKS	1 DAY	
	1 DAY				8 WEEKS	5 DAYS	

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**)

The screenshot shows a CI status page with a summary message: "All checks have passed" followed by "9 successful checks". Below this, there is a list of five build steps, each with a green checkmark icon, indicating success. 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)". Each step has a "Details" link next to it.

The screenshot shows a GitHub pull request interface. At the top, it says "Tools: extract_features.py: correct define name for AP_RPM_ENABLED" and "peterbarker committed 5 days ago". Below this, there are three more commit entries: "AP_Mission: prevent use of uninitialised stack data" by "peterbarker" (5 days ago), "AP_HAL_ChibiOS: disable DMA on I2C on bdshot boards to free up DMA ch..." by "andy1per" and "tridge" (6 days ago), and "SITL: Fixed rounding lat/lng issue when running JSBSim SITL" by "ShivKhanna" and "tridge" (6 days ago). At the bottom, it shows another commit: "AP_HAL_ChibiOS: define skyviper short board names" by "yuri-rage" and "tridge" (6 days ago). There are also two small red 'X' icons next to some of the commit descriptions.

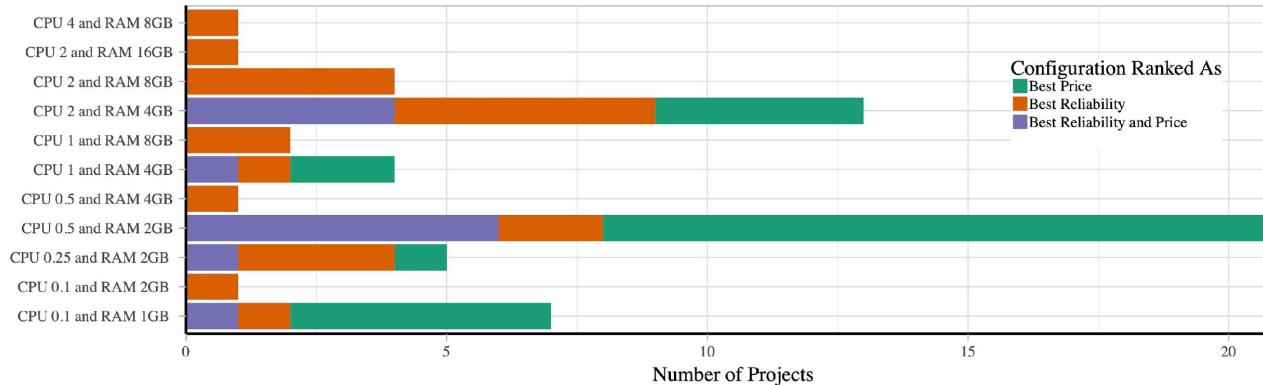
Effective CI processes are run often enough to reduce debugging effort

- 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 Passing
✓ 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

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

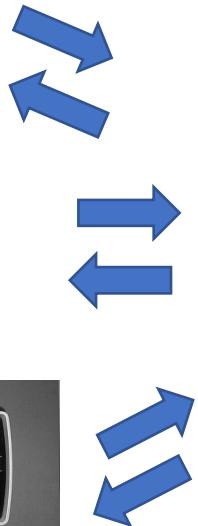


Photo Credit: ArnoldReinhold, CC BY-SA 3.0 via Wikimedia Commons



Photo Credit:
[Wikipedia](#)

1980s & 1990s Personal Computing

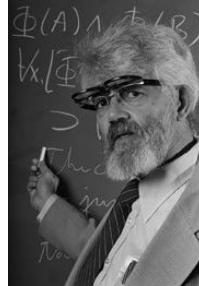
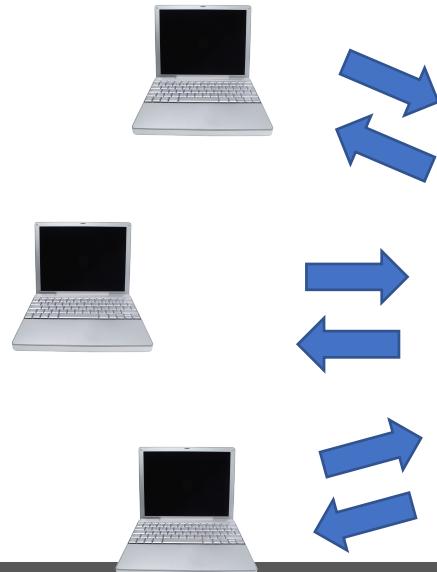


Photo Credit: Rama & Musée Bolo, [CC BY-SA 2.0 FR](#) via Wikimedia Commons



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

2000s Cloud Computing

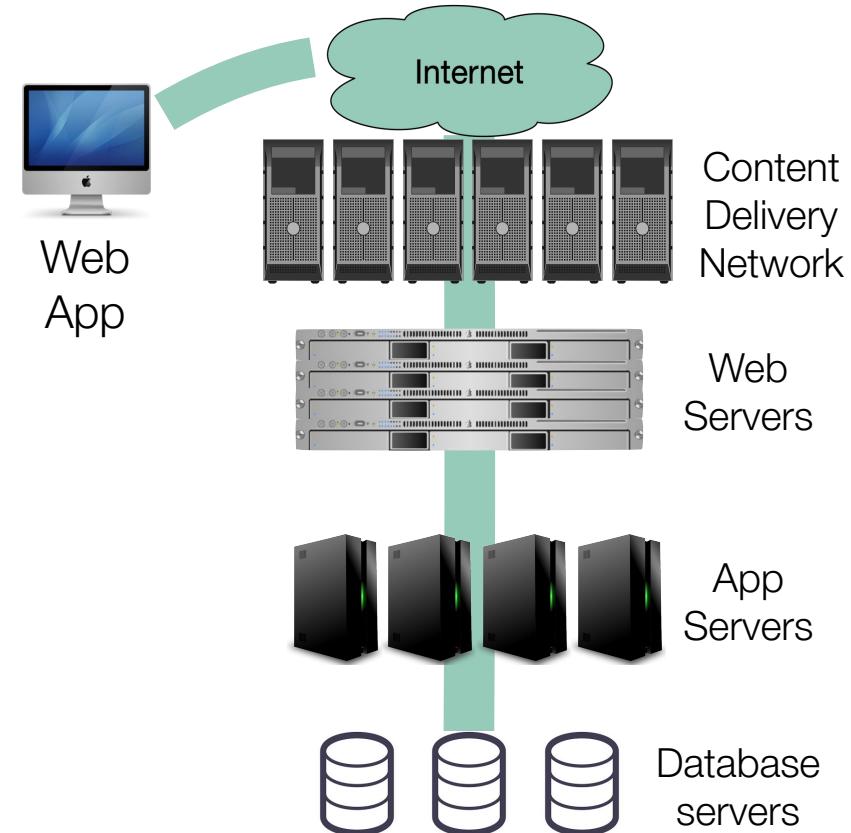


"Computing may someday be organized as a public utility just as the telephone system is a public utility...Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system ..."

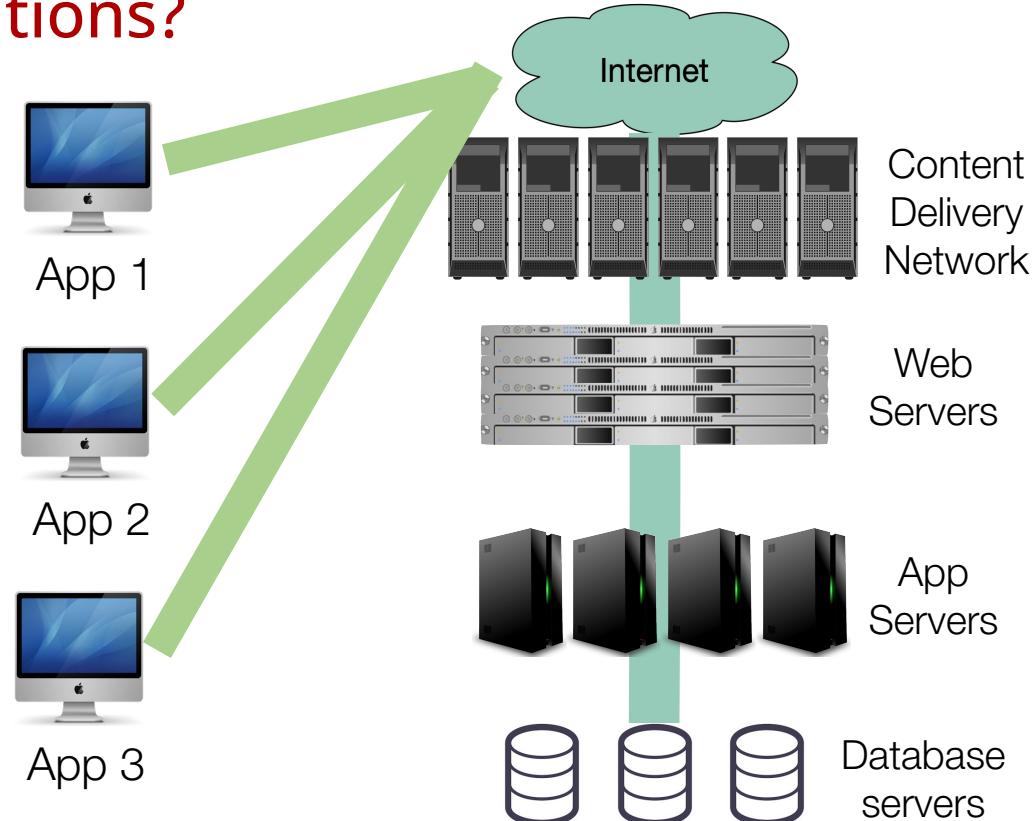
McCarthy's predictions come true!

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?



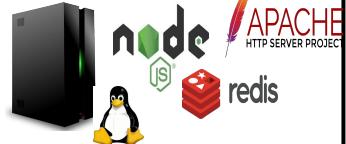
Multi-Tenancy creates economies of scale

- 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

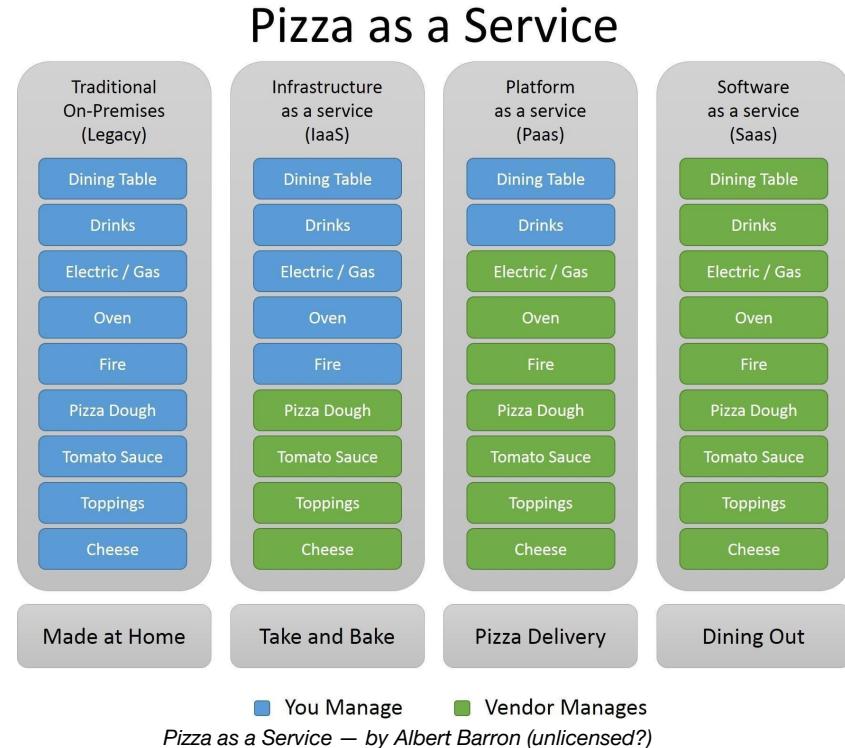
- “Traditional” computing infrastructure requires capital investment
 - “Scaling up” means buying more hardware, or maintaining excess capacity for when scale is needed
 - “Scaling down” means selling hardware, or powering it off
- Cloud computing scales elastically:
 - “Scaling up” means allocating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

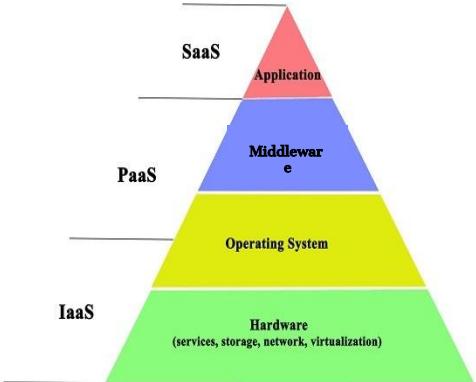
Cloud Computing: Analogy using NodeBB

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

Shared infrastructure analogy: Pizza

- Four ways to get pizza: Make yourself, take and bake, delivery, dine out
- Vendor manages different levels of the stack, achieving economies of scale
- When would you choose one over the other?





Activity

Pick one scenario based on where you are seating

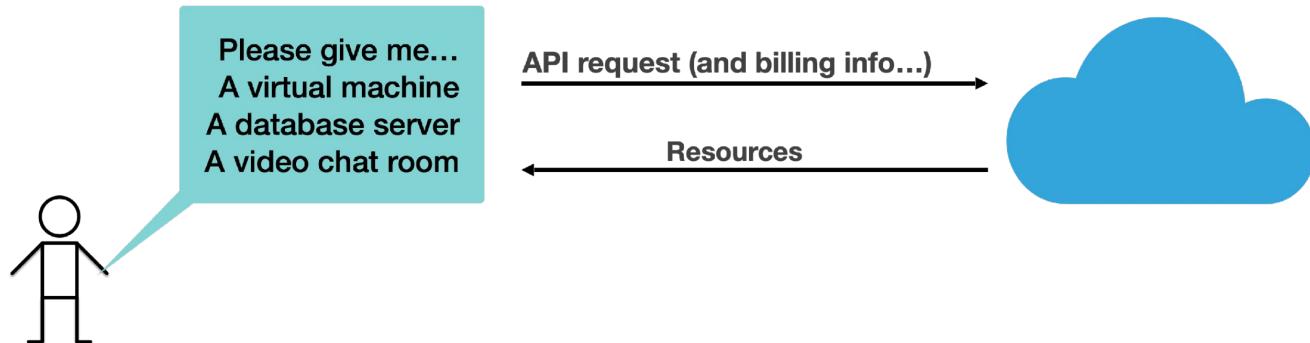
- Software as a Service - SaaS (front rows)
- Platform as a Service - PaaS (middle rows)
- Infrastructure as a Service - IaaS (back rows)

Discuss in groups of 2-3 the applicability of the assigned cloud service model (IaaS, PaaS, or SaaS)

- Brainstorm and come up with at least **two** real-world scenarios 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.

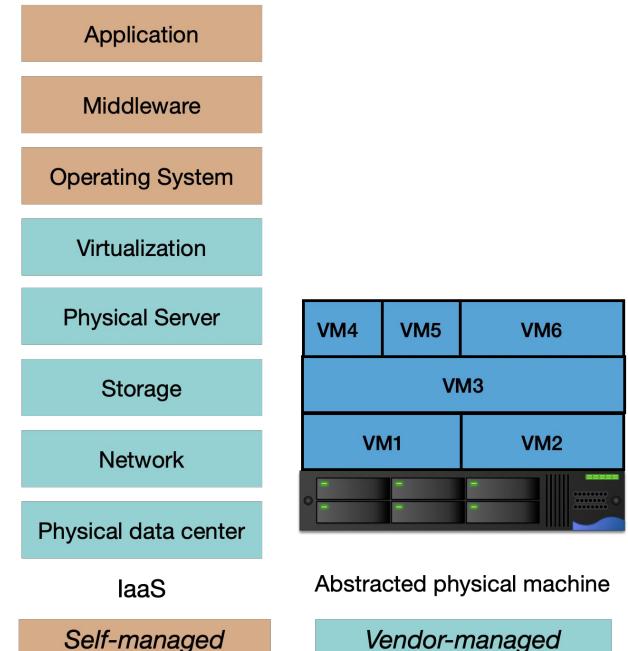
Cloud services gives on-demand access to infrastructure, “as a service”

- 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

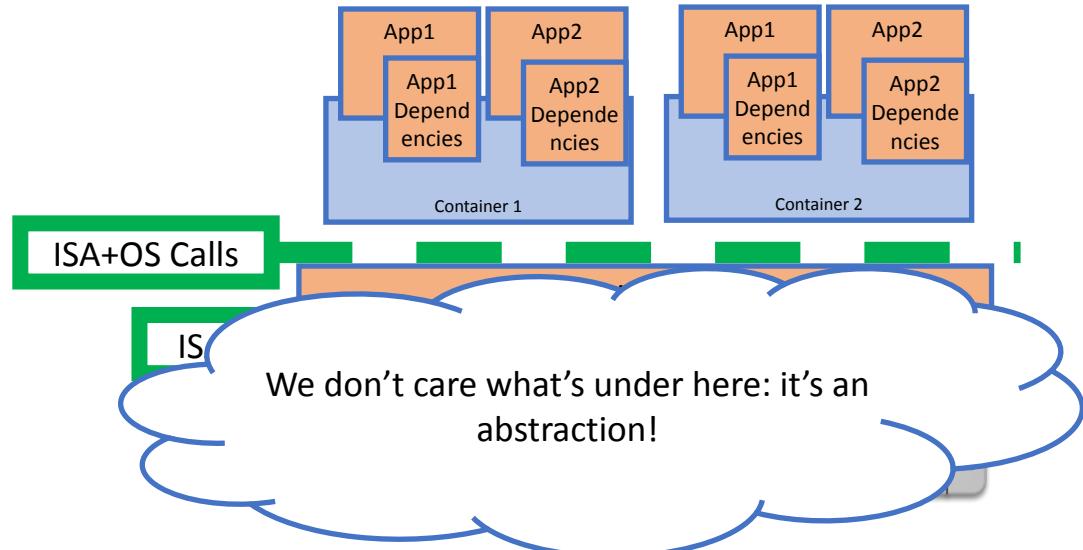


Virtual Machines to Containers

- 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)

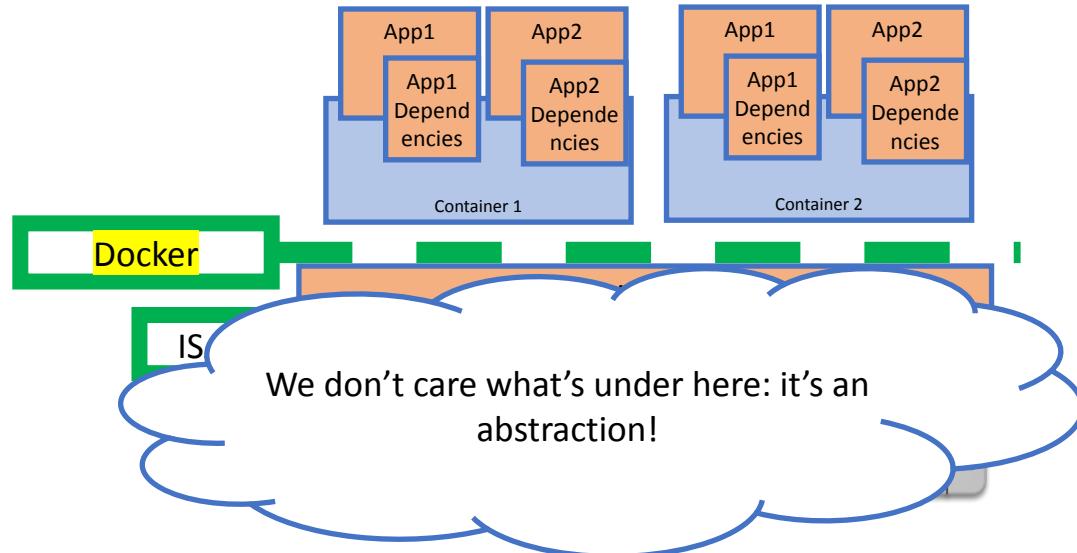
CaaS: Containers as a Service

- 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 ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

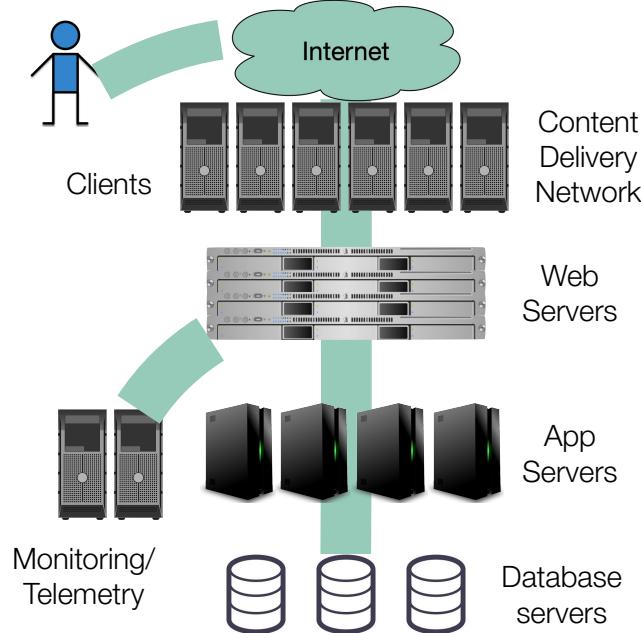
```
1  FROM node:lts
2
3  RUN mkdir -p /usr/src/app && \
4      chown -R node:node /usr/src/app
5  WORKDIR /usr/src/app
6
7  ARG NODE_ENV
8  ENV NODE_ENV $NODE_ENV
9
10 COPY --chown=node:node install/package.json /usr/src/app/package.json
11
12 USER node
13
14 RUN npm install --only=prod && \
15    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

Tradeoffs between VMs and Containers

- Performance is comparable
- Each VM has a copy of the OS and libraries
 - Higher resource overhead
 - Slower to provision
 - Support for wider variety of OS'
- Containers are “lightweight”
 - Lower resource overhead
 - Faster to provision
 - Potential for compatibility issues, especially with older software

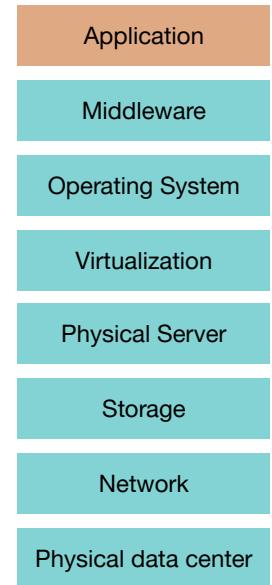
Platform-as-a-Service: vendor supplies OS + middleware

- 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



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