# CS45, Lecture 16
# Cloud & Serverless

**Winter 2023**

Akshay Srivatsan, Ayelet Drazen, Jonathan Kula

# Learning Goals

- Understand what “cloud computing” really means
- Understand what “serverless computing” means and when to use it
- Understand the benefits and limitations of each (and versus self-hosting), and know when to use one or the other
- **Spin up your first cloud computer and ssh into it**
- **Deploy your first serverless function**
- ...and do both of those for free ;)
- Know about a few different providers for both cloud and serverless computing

# Context: Hosting some service

Most of this lecture will be contextualized in terms of **running an application or service on a remote machine.**

For example:

- Running a server (that you want others to be able to access)
- Running some kind of application that does something for you periodically
- Storing or syncing your files
- etc.

# What's a "cloud"?

*"The Cloud is just someone else's computer"*

At its most basic: **a cloud is a server.** Often, it is a cluster of servers.

In popular tech nomenclature, a "cloud" often refers to a conglomerate of interconnected services offered by a particular provider, e.g. AWS or Google Cloud

- "The AWS cloud"
- "The Google Cloud"
- etc

# Benefits of a “cloud”

Why use someone else’s computer?

- They manage the computer for you
  - Usually resistant or reimbursed for downtime
  - Always on (as long as you want)
  - Usually don’t have to worry as much about outages
- Cost is amortized over time
  - $2000 gaming computer + maintenance vs $0.50 per hour
  - Only pay for what you use (on-hours)
- They offer services you don’t have access to
  - Hardware encryption modules
  - Enterprise networking setups
  - Serverless functions
- They abstract away complex services
  - Caching
  - DNS
  - Kubernetes
  - Databases
  - more...

# Detriments/a cautionary tale

Why not use someone else's computer?

- Cost
  - If you're just doing stuff as a hobby, you may be able to use resources available to you for free
    - Old or cheap computers
  - If you're not careful, you can end up being charged thousands
    - "Forgot to turn off my AWS instance" = $100s+ down the drain
- Direct access to hardware
  - You have to use the cloud provider's abstractions
- Very specific setups
  - E.g. if you need something on your local network, or connected to specialized hardware
    - E.g. 3D printers
- Fun?
  - I've had a lot of fun running my own servers for free on old hardware

# An “Instance” / “Server”

**Enterprise Clouds can do a lot of specialized stuff!**

*But in this lecture we’re going to do something powerful, but basic:*

- Run our own **compute instance** i.e. **our own special server!**

We’ll connect to this server using **ssh** (“Secure Shell”)

# Demo: Launching your first instance

High level outline:

- “Instances” in Oracle Cloud
- New Instance
- Free tier, with Ubuntu Server
- Add SSH public key for authentication
- Provision (and wait)

# Administering Our First Server

We'll be running **Ubuntu Server** (a distribution, or "distro" of Linux).

In order to use it, we'll have to do some **system administration** tasks!

- For us, at a basic level: **Installing packages**

In Ubuntu, we do this using `apt`, the Advanced Packaging Tool

# Demo: Installing Docker & Python

High level overview:

- SSH into our new instance
- Install Docker
  - `curl -fsSL https://get.docker.com | bash`
- Install python3
  - `sudo apt install python3`

# Demo: Running Calculator via Docker

High level overview:

- Use newly-installed Docker to run calculator image I published

# Demo: Running Our Basic Python Server

High-level overview:

- Use `python3 -m http.server 8080` to start a server as before
- Attempt (and probably fail) to access it

# Demo: Opening Ports on Oracle

High level overview:

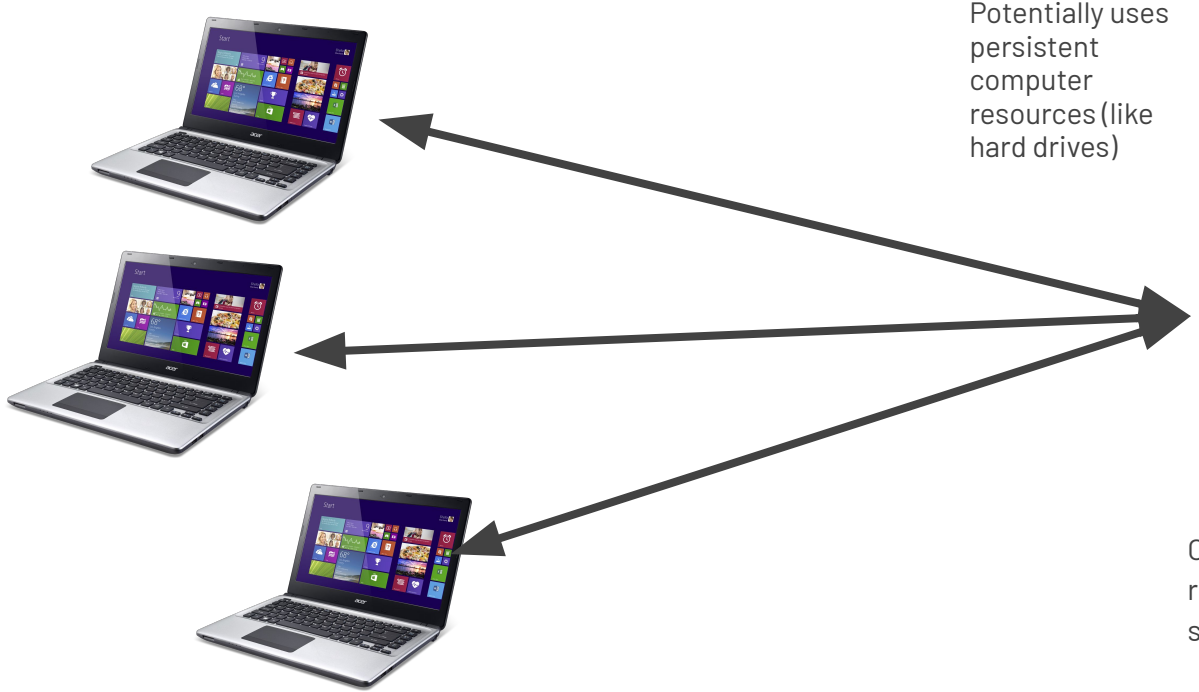
- Go to subnet security settings and add a rule to:
  - allow TCP
  - on destination port 8080
  - From 0.0.0.0/0
- Then let’s try accessing our server again!

# Various Cloud Providers

- AWS (Amazon Web Services)
- GCP (Google Cloud Platform)
- Microsoft Azure
- Oracle Cloud
- DigitalOcean

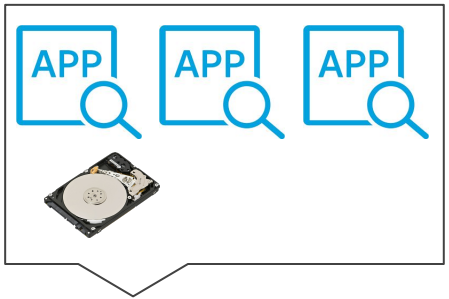
# Server vs. Serverless

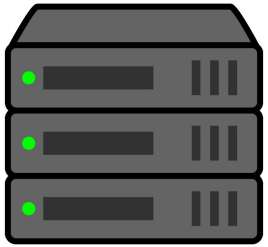
Traditional servers work like this:



Potentially uses persistent computer resources (like hard drives)

Inside: many different apps potentially working in tandem



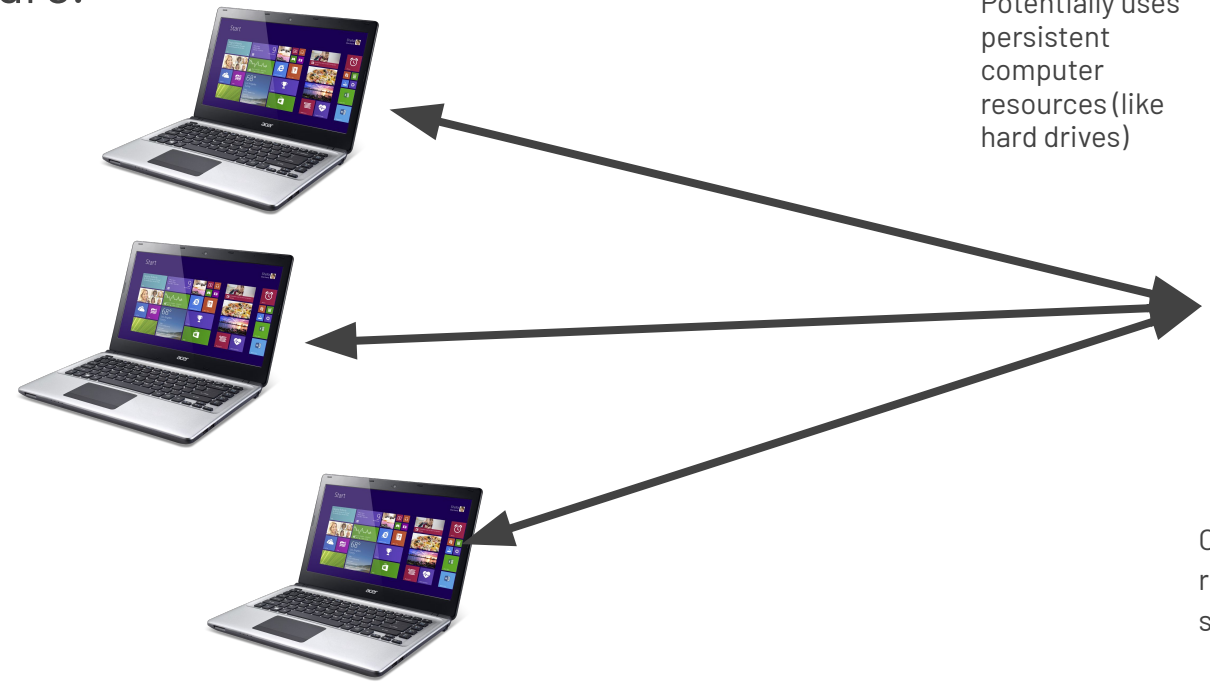





One single (or a cluster of) recognizable, consistent servers. **Manually configured.**

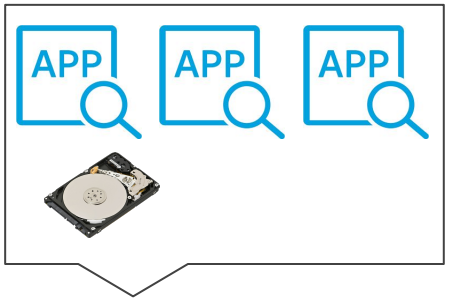
# Server vs. Serverless

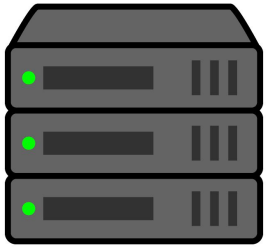
This can end up bloated, or with single points of failure.



Potentially uses persistent computer resources (like hard drives)

Inside: many different apps potentially working in tandem



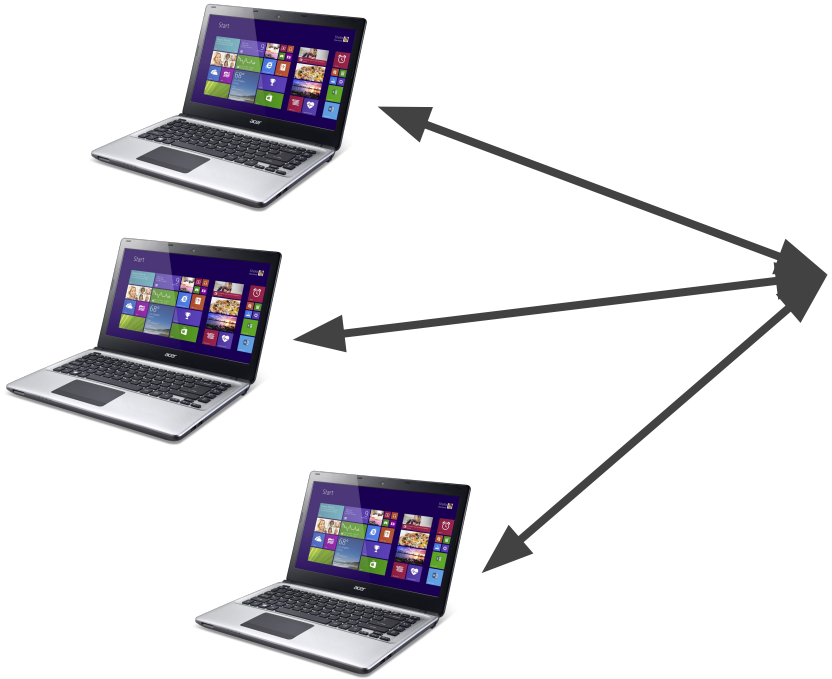





One single (or a cluster of) recognizable, consistent servers. **Manually configured.**

# A Less Centralized Approach

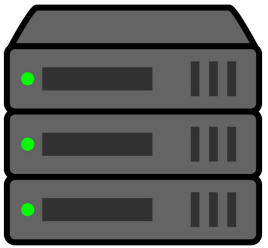
Instead, we can imagine decoupling everything.




**Functions** (not full applications!) – parts of your app that can run **independently**

^^Literally any computer connected to Amazon's network

Inside: only applications that need persistent storage



One single (or a cluster of) recognizable, consistent servers (backend only)

# Servers vs Serverless

- Traditional servers run **software**.
  - → Usually the software **directly interacts with the network** (if it talks to the network)
  - → Requires **configuration** and **server administration**
  - → Software is **always running** to respond to events
- Traditional servers **run consistently on the same (set of) machine(s)**

- Serverless systems run **functions**.
  - → The functions have a defined language, input, and output.
  - → Functions are **only invoked when needed** (e.g. on request)
  - → The software needs to be **written to take advantage of that system**
  - → Aside from that, **very little configuration**
- Serverless functions **are ephemeral**
  - → short-lived (max runtime ~10 seconds)
  - → cannot store files/etc
- Serverless functions **run anywhere** (no consistent machine).
  - There are still servers involved, you just don't know which ones/the scale is "infinite"
  - Usually charged per invocation

# Servers vs Serverless

## Good tasks for serverless

- Serve a static webpage
  - Webpage is built into the code; no persistence required
- Generate a webpage from a user's cookies
  - Cookies are included with the request, no need to store anything
- Retrieve and serve information from an external database
  - Database is located somewhere else, so no need for persistence

# Servers vs Serverless

**Bad tasks for serverless (probably need a server)**

- Do something periodically
  - Serverless only runs on request
- Run a database
  - Serverless functions don't persist data; you'd need a server with persistent storage (e.g. a hard drive)

**Depends**

- Receive user uploads
  - Serverless can receive them, but they would have to be then re-uploaded to an external service (e.g. AWS S3)

# Serverless <3 Servers

- Often **servers + serverless will work together!**

Here's an example from an app I built:

- Use a **server** to host a database
- Use **serverless functions** to host a website
- Use **serverless functions** to provide APIs to request downloading music
  - Adds a download request to the database
- Use a **server** to download, encode, and upload media (using ffmpeg)
- Use **serverless functions** to retrieve information from the database, (like where the music ended up).

# Demo: Deploy a serverless function

High level overview:

- Introduce Vercel
- Read their documentation about how to create a serverless function
- Write a simple request handler in NodeJS
- Deploy it on Vercel

# Various Serverless Providers

*Built on top of/as a part of cloud platforms:*

- AWS Lambda
- GCP Serverless
- Azure Serverless
- DigitalOcean Functions

*Their own services (usually designed for website hosting-esque tasks):*

- Vercel (runs on AWS Lambda)
- Netlify
- Cloudflare