THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Cloud and Web Apps

CSCI 2541 Database Systems & Team Projects

Wood & Chaufournier

Slides adapted from Prof. Bhagi Narahari; and Silberschatz, Korth, and Sudarshan

Phase 2 has begun!

You should have an email / slack message about your phase 2 assignment

Builder: build REGS or APPS (whichever you were assigned)

- APPS: No PhD apps or Sys Admin role
- REGS: No user creation, no PhD students, no Sys Admin

Integration: Combine and improve 2-3 components

- 2 team members -> 2 components
- 3 team members -> 3 components
- REGS is the bridge!

Timeline

Today: Web app architectures

Wednesday April 21:

- Web Services Lab
- Mentor meetings

Monday April 26

- Course Summary / wrap up

Wednesday April 28 (GW "Monday")

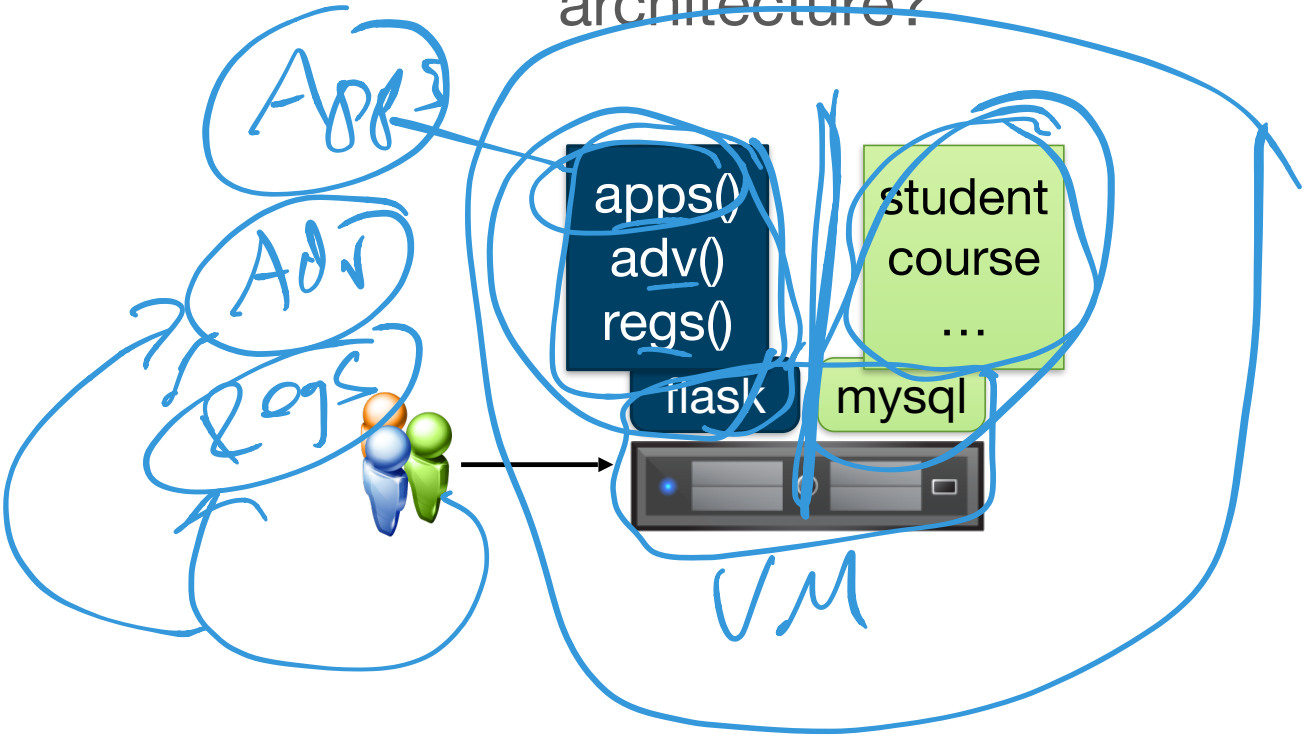
- Mentor meetings

Monday May 4 - Tuesday May 11: Schedule an appointment to perform final demo

Phase 2 questions?

Your Project

What other ways are there to design this architecture?



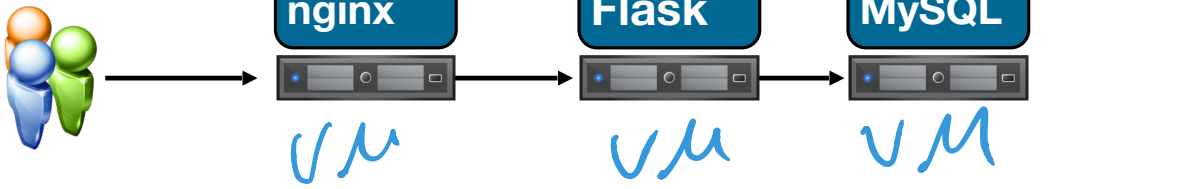
Multi-Tier Web Applications

Traditionally composed of 3 components

Separation of duties:

- **Front-end web server** for static content (Apache, lighttpd, nginx)
- **Application (API) tier** for dynamic logic (PHP, Flask, node.js)
- **Database back-end** holds state (MySQL, MongoDB, Postgres)

Why divide up in this way?



Stateful vs Stateless

The multi-tier architecture is based largely around whether a tier needs to worry about state

Front-end - totally **stateless**

- There is no data that must be maintained by the server to handle subsequent requests

Application tier - maintains **per-connection state**

- There is some temporary data related to each user, e.g., my shopping cart
- May not be critical for reliability - might just store in memory

Database tier - **global state**

- Maintains the global data that application tier might need
- Persists state and ensures it is consistent

N-Tier Web Applications

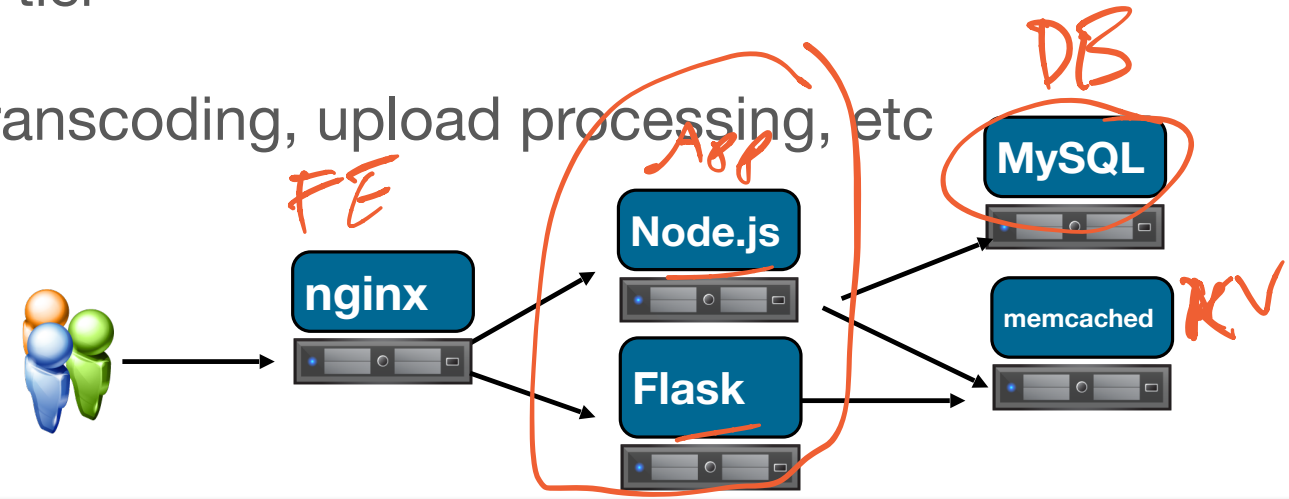
Sometimes 3 tiers isn't quite right

Database is often a bottleneck

- Add a cache! (stateful, but not persistent)

Authentication or other security services could be another tier

Video transcoding, upload processing, etc

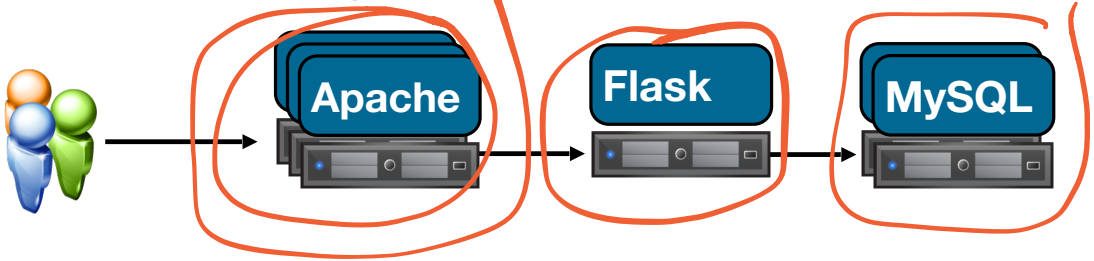


Replicated N-Tier

Replicate the portions of the system that are likely to become overloaded

How easy to scale…?

- Apache serving static content
- Flask application managing user shopping carts
- MySQL cluster storing products and completed orders

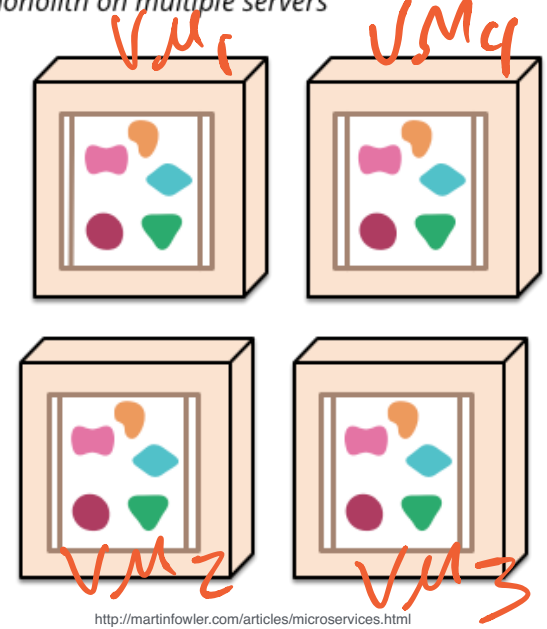


Tune number of replicas based on demand at each tier

Application Tier

A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers



http://martinfowler.com/articles/microservices.html

Monolith: One piece of software that contains the full functionality of the application

Problems with **monolithic** approach?

Monolithic Challenges

Scalability: Need to possibly use both up and out to reach performance goals. Hard to scale things like databases.

Reliability: A fault/memory leak in a single place can crash the entire app.

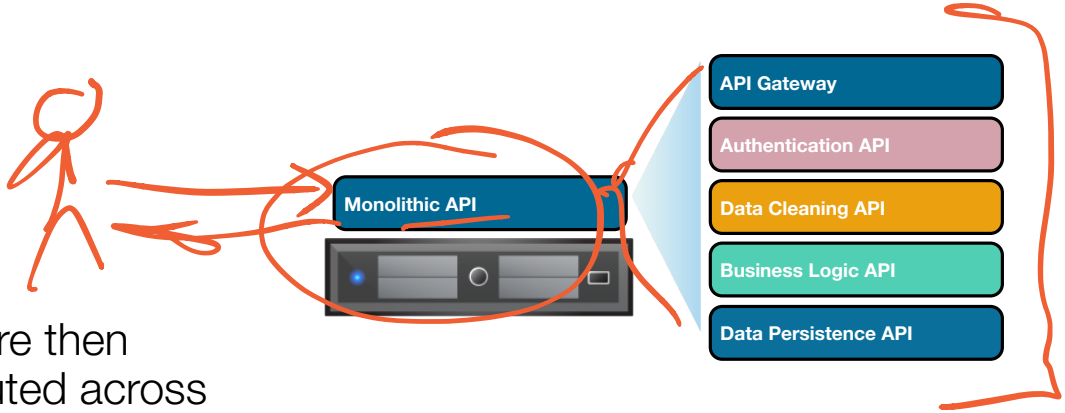
Orchestration: You need to rebuild and deploy the entire application every time you make a change.

Code Complexity: Code turns into spaghetti due to too many things happening at once. Hard to refactor features.

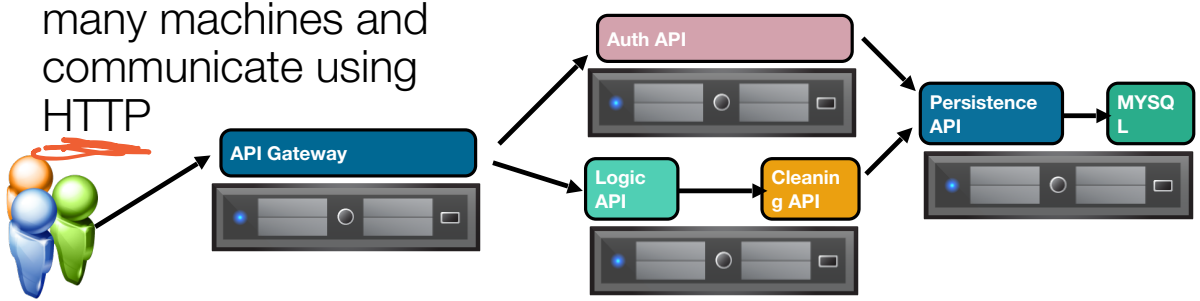
Upgradeability: Moving to newer tech stacks requires converting the entire app at once.

Microservices

Take your application API and split it into smaller components based on function.



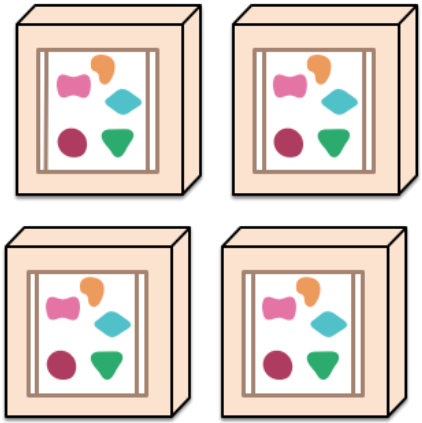
API's are then distributed across many machines and communicate using HTTP



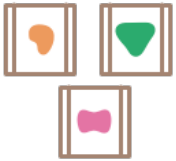
Microservices

A monolithic application puts all its functionality into a single process...

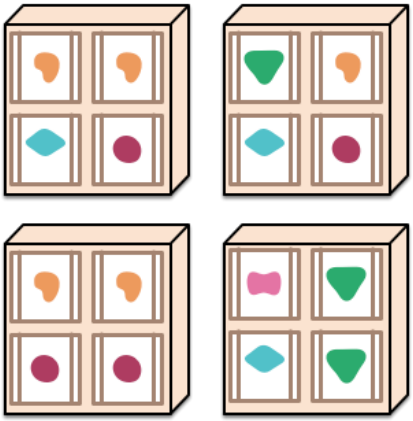
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.

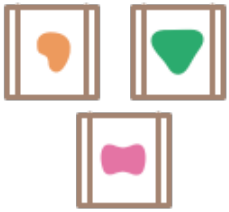


Read more: https://martinfowler.com/articles/microservices.html

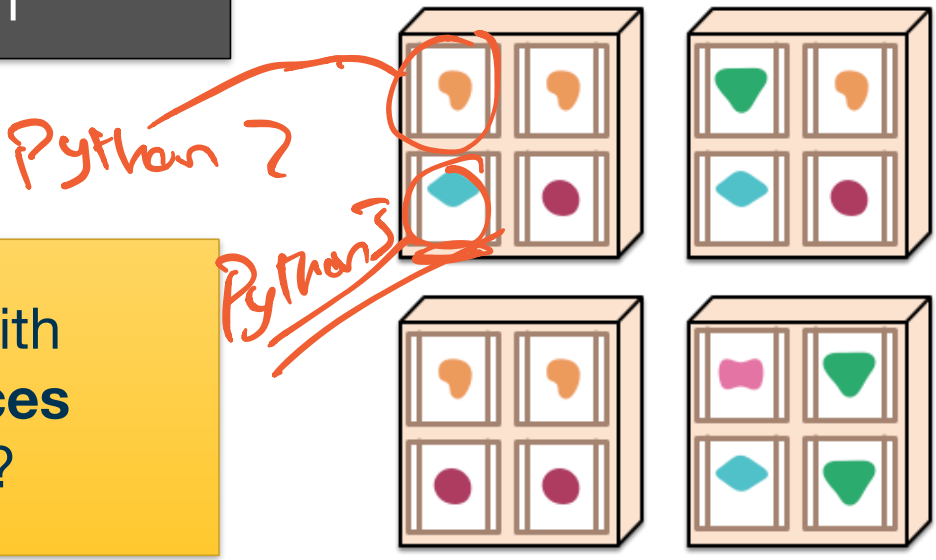
Microservices

Microservice: A small piece of functionality which can be composed with others to build an application

A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Problems with **microservices** approach?

Microservices Challenges

Discovery: how to find a service you want?

Scalability: how to replicate services for speed?

Openness: how to agree on a message protocol?

Fault tolerance: how to handle failed services?

All distributed systems face these challenges, microservices just increases the scale and diversity…

Netflix

20th most popular website according to Alexa

Zero of their own servers

- All infrastructure is on AWS (2016-2018)
- Recently starting to build out their own Content Delivery Network

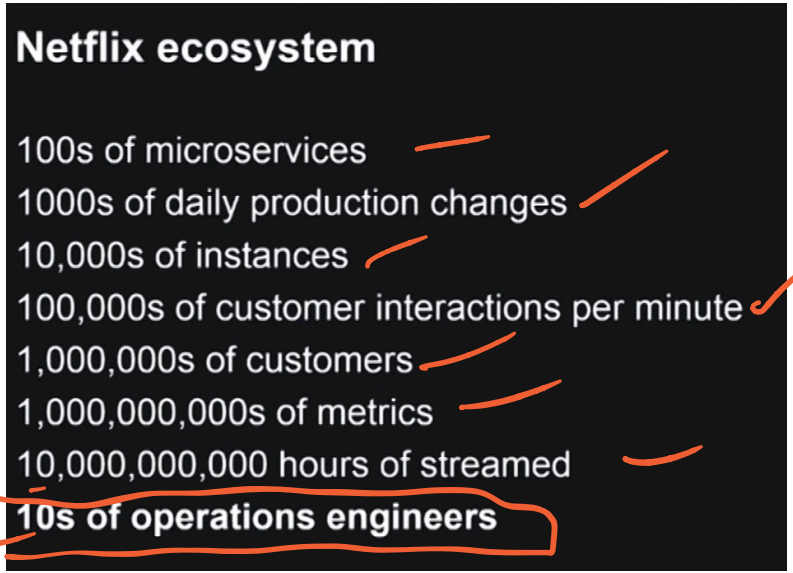
Netflix

One of the first to really push microservices

- Known for their DevOps
- Fast paced, frequent updates, must always be available

700+ microservices

Deployed across 10,000s of VMs and containers



Netflix tech talk: https://www.youtube.com/watch?v=CZ3wIuvmHeM

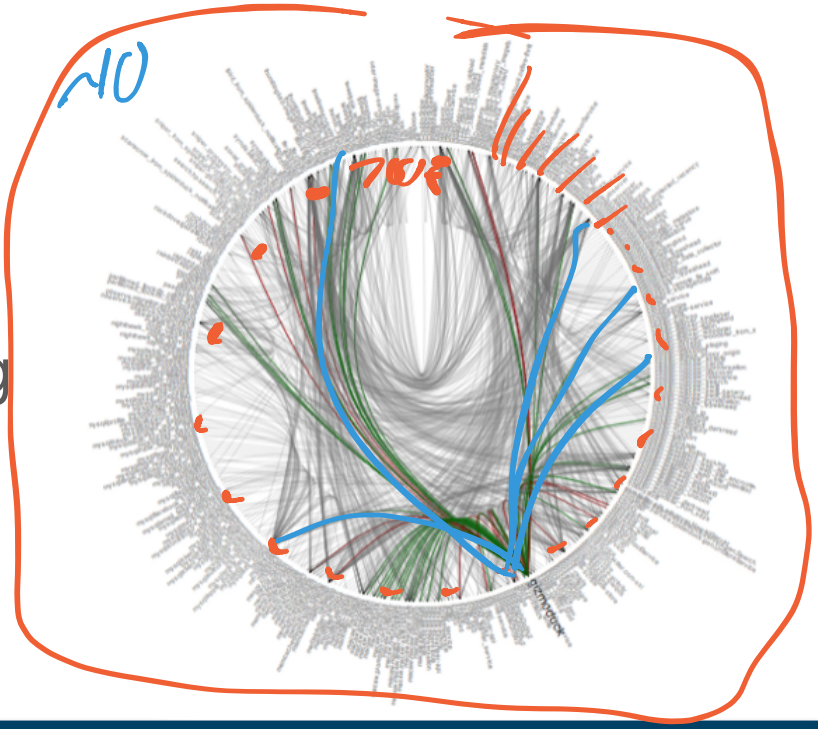
Netflix "Deathstar"

Microservice architecture results in a **extremely** distributed application

- Can be very difficult to manage and understand how it is working at scale

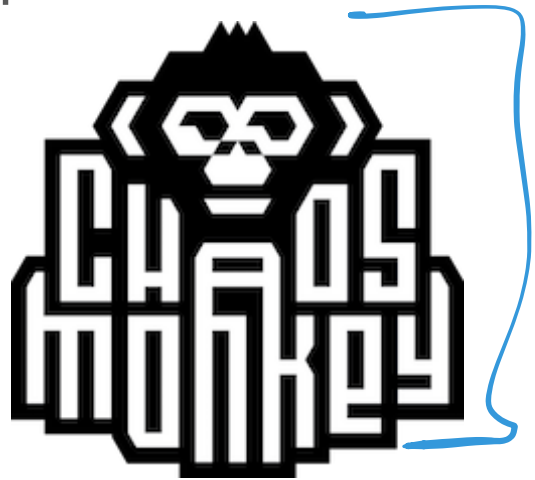
What if there are failures?

How to know if everything is working correctly?



Netflix Chaos Monkey

Idea: If my system can handle failures, then I don't need to know exactly how all the pieces themselves interact!



Chaos Monkey:

- Randomly terminate VMs and containers in the production environment
- Ensure that the overall system keeps operating
- Run this 24/7

Make failures the common case, not an unknown!

http://principlesofchaos.org/

Team talk

Any more general Phase 2 questions?

Go to your team # breakout room

Call for help if you want to talk to a mentor

We will drop by a few rooms to check status

You can leave at end of period even if we haven't visited you

Get started! Be efficient! Communicate!