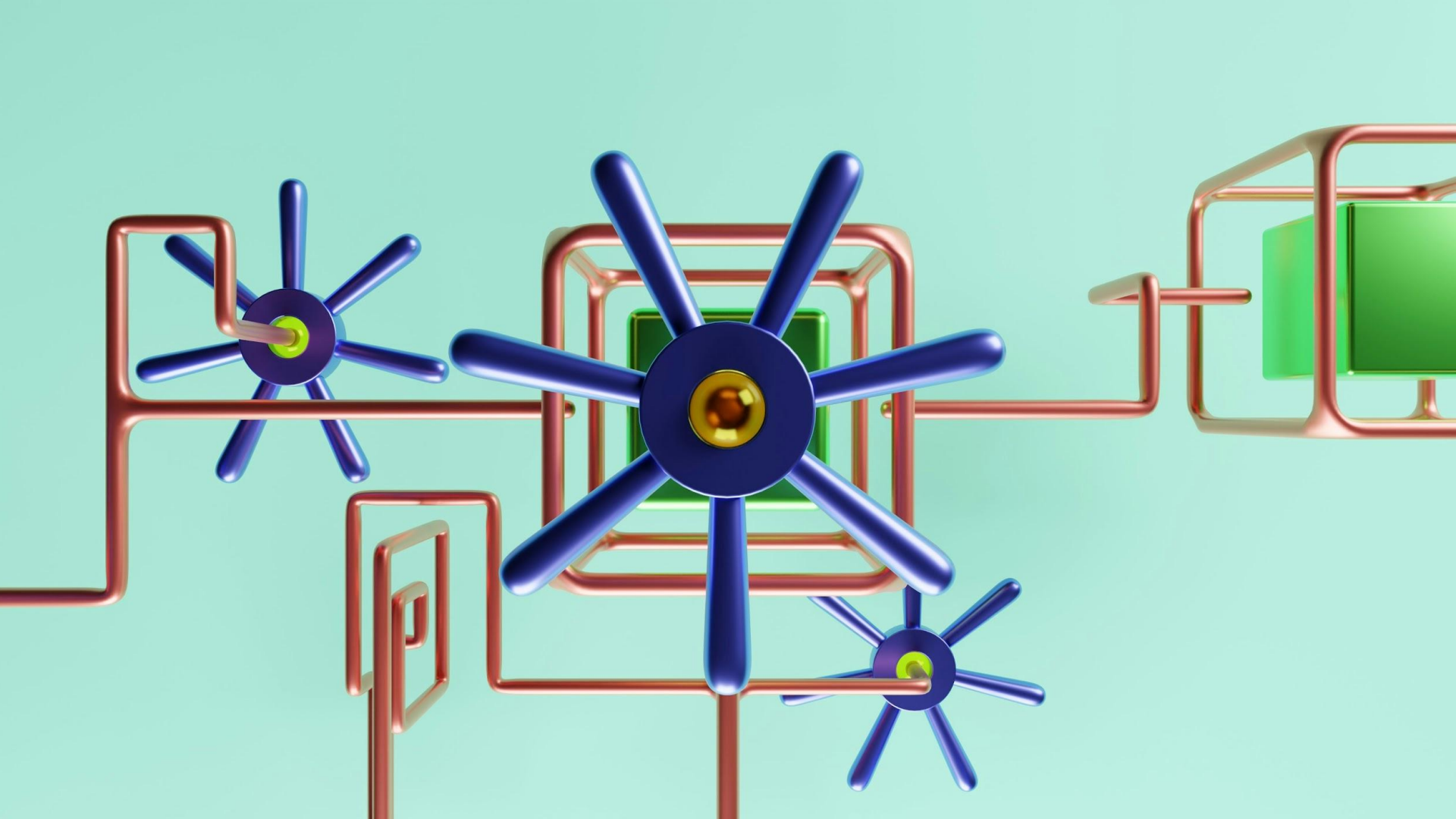
Grand Valley State University
www.gvsu.edu

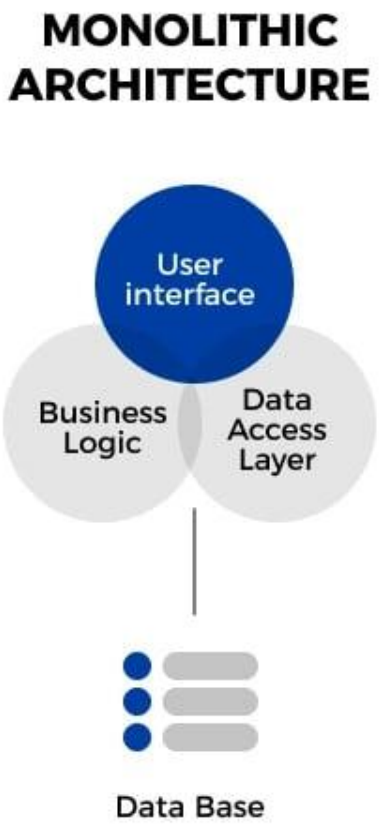
# Cloud Computing Microservices

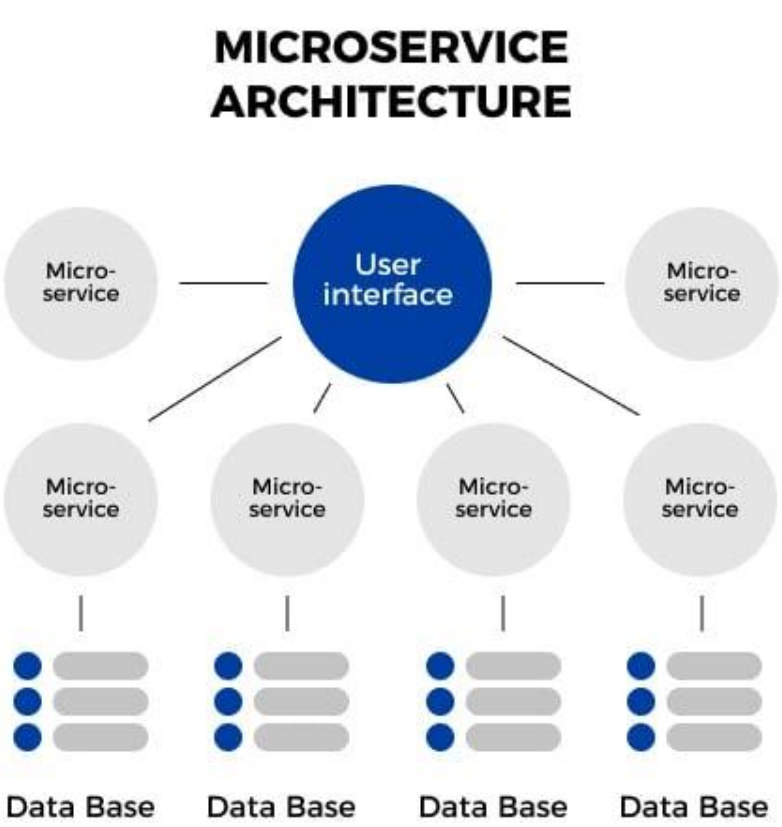
CIS437
Erik Fredericks // frederer@gvsu.edu

*Adapted from Google Cloud Computing Foundations, Overview of Cloud Computing (Wufka & Canonico)*

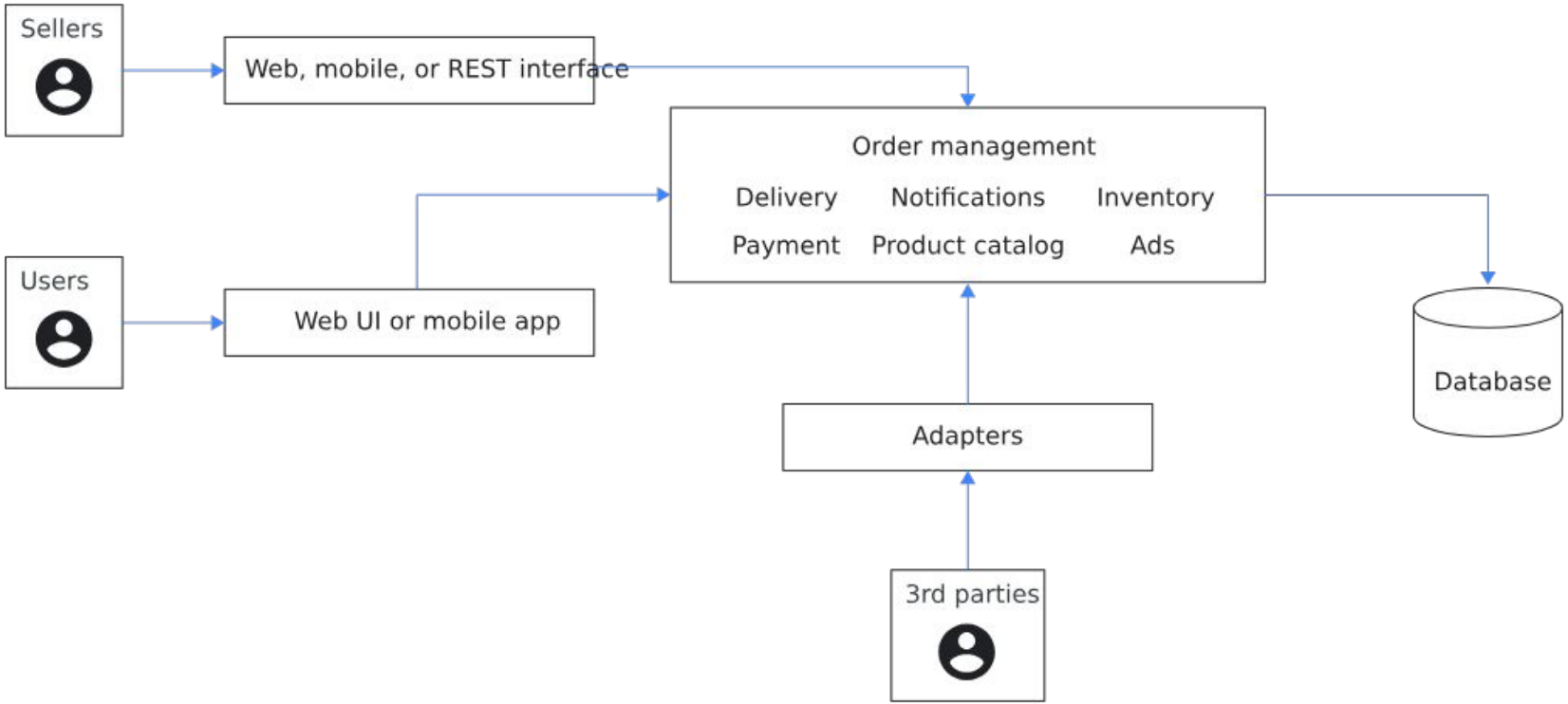


https://www.openlegacy.com/blog/monolithic-application
https://cloud.google.com/architecture/microservices-architecture-refactoring-monoliths

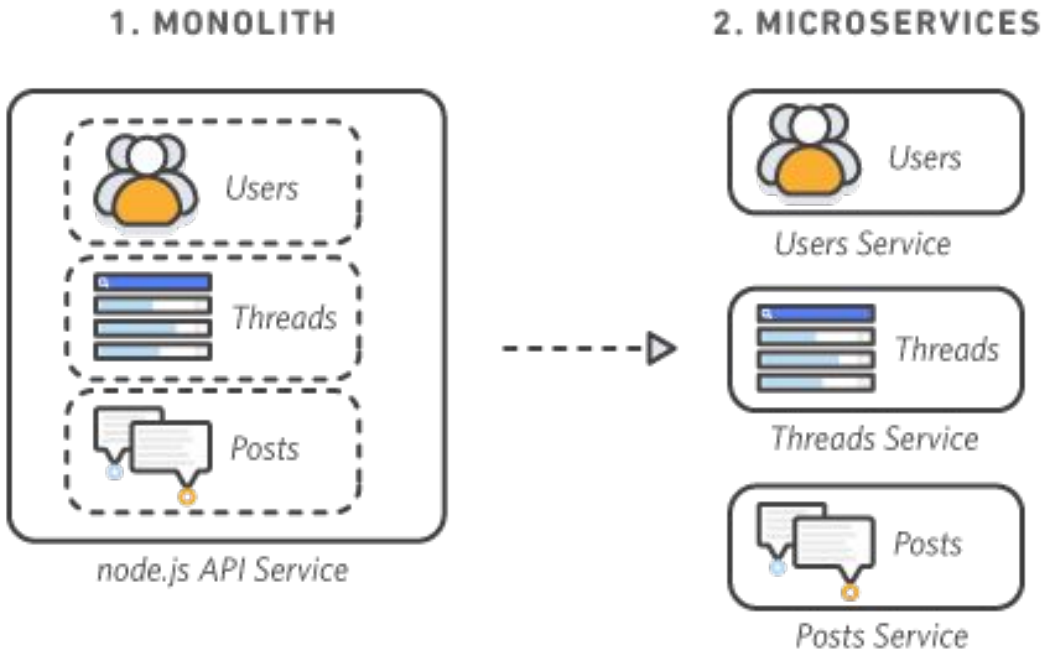


# Example: a monolith design



https://aws.amazon.com/microservices/



# What is a *monolithic* application?

Typically a single entity that controls everything

- "one big app"
    - single codebase, single program, etc.

Obviously could mean many different things - in the cloud space we see it as a networked application with all moving parts in one project

- Front end (user interface)
- Back end (server)
- Database (...database)

# Problems with this?

Quite nice to have everything in one place, **but**:

- Updates become problematic
  - **Everything** needs to be re-validated/checked
  - Apps tend to grow over time
- Scalability is a problem
  - How do we offload a single app to another server easily?
- Doesn't easily port to the cloud!
  - Lots of updates necessary to get it up and working with cloud services

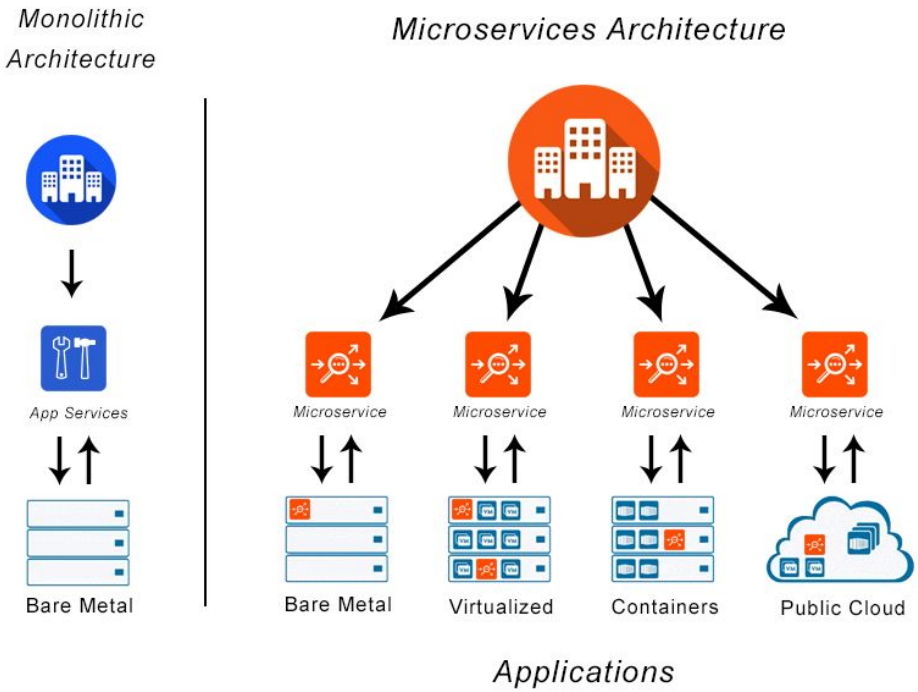
# Enter microservices

Single app from multiple smaller apps
- Typically connected via networks

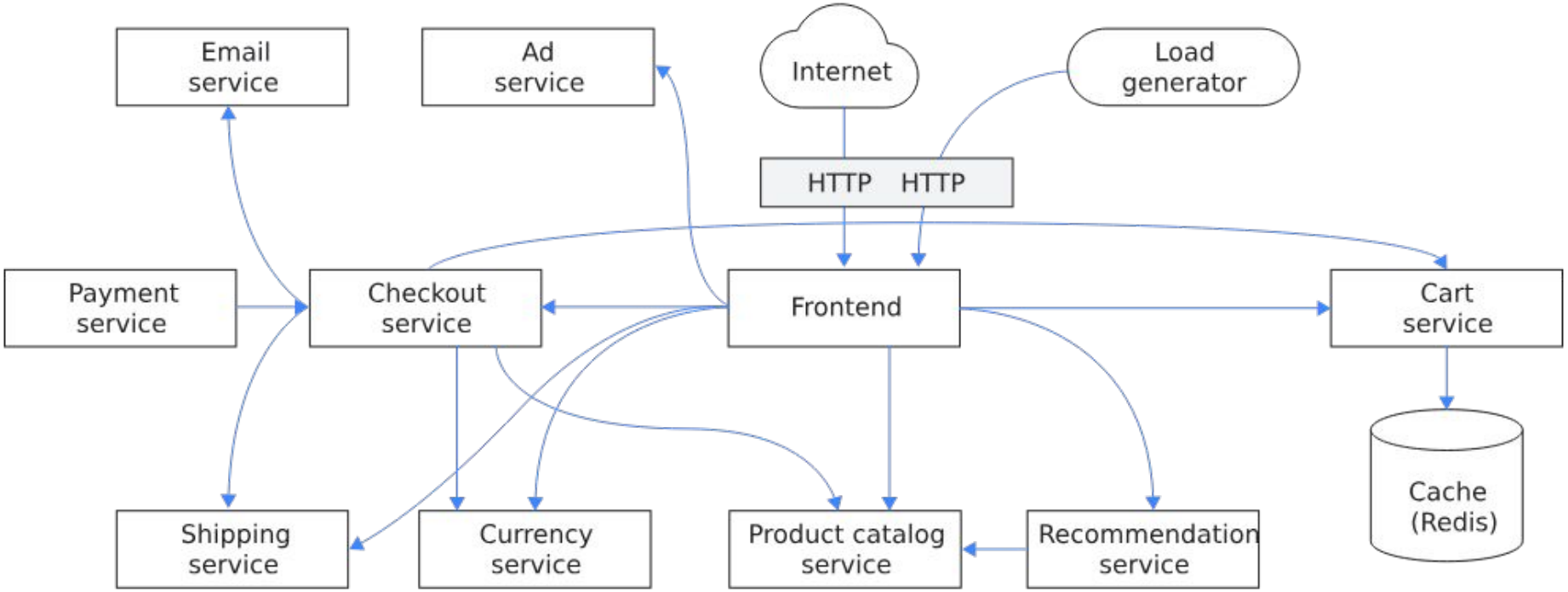
Loosely coupled
- (Not a lot of overlap)

Independently deployable // good for a CI/CD pipeline

In theory, a good way to build a scalable architecture



# Monolith to microservices



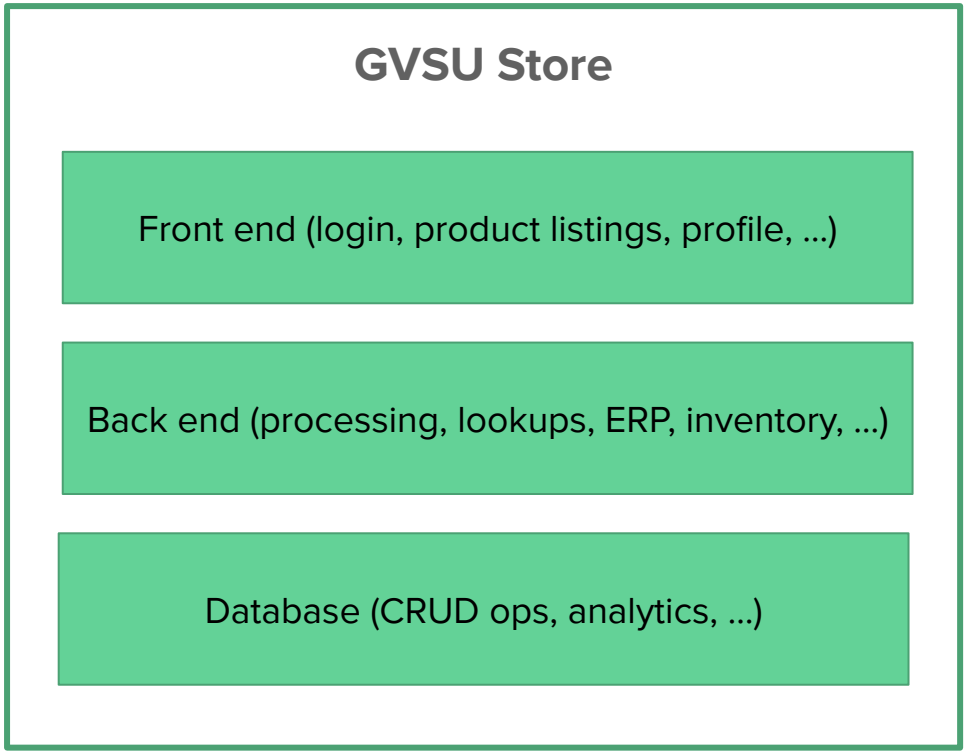
# An example!

Start with a large legacy monolithic application

- For example, an online store

Everything is one big happy React app

How do we go about turning it into a collection of microservices?



# Arguments for microservices

Separation of concerns within a large application

Allows for independent teams to work simultaneously
- CI/CD pipelines can be a real help here!

Scaling happens much easier here
- Each microservice can deploy with any number of instances necessary
  - And can scale up further as well!
- Constraints are limited to the microservice itself, not the app as a whole

Updates are partitioned to their own microservices
- Don't *necessarily* impact application as a whole

# Arguments against microservices

Each service must be well-defined

- APIs, network calls, etc.
- What happens if its IP address changes?

Security a concern

- Instead of 1 attack vector, now there are *n* attack vectors

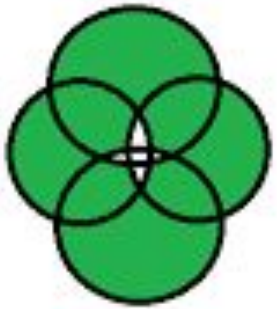
$$$

- Each microservice induces a charge
- What happens if one gets called *way* more than it should be?
- Or, stores too much data?

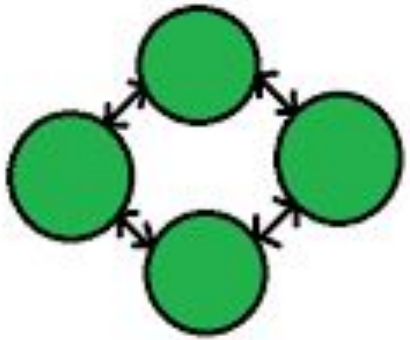
# Regardless...

Decoupling services can be a good idea

- Pretty commonplace suggestion in software engineering
  - Why not in cloud?



Tight coupling:
1. More interdependency
2. More coordination
3. More information flow

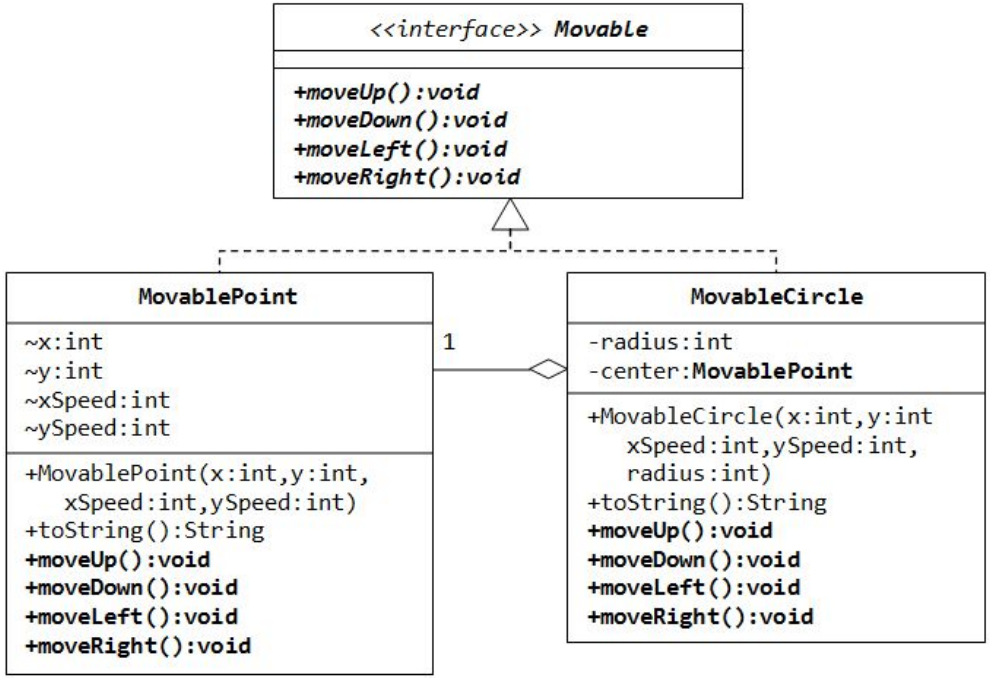


Loose coupling:
1. Less interdependency
2. Less coordination
3. Less information flow

# Decoupling

How can we decouple programs (in general)?

- Single-purpose classes (laser focused)
- Reduce interdependencies
- Reduce redundancies
  - Why?



# Decoupling (☁️ version)

**Review existing documentation**

Determine what can be broken apart
- Then determine how it can be stitched back together

Create **well-defined** specifications
- i.e., your API
- How does data flow from app to app?
- How can things be *compartmentalized*
- What are the implications of doing this as well?
    - Money a concern here?

# Domain-driven design (DDD)

https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design

https://www.geeksforgeeks.org/domain-driven-design-ddd/

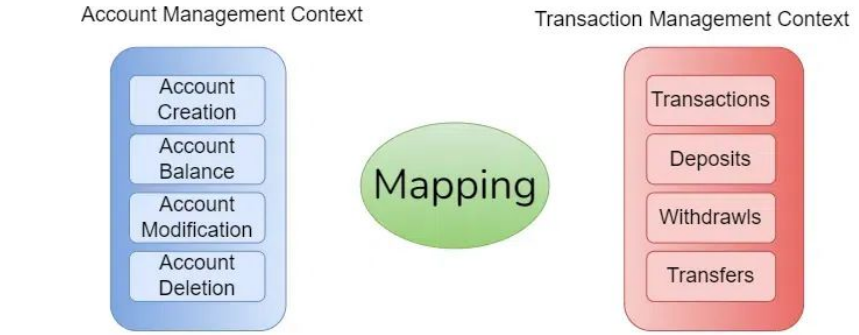
"understanding and modeling the problem domain within which a software system operates"

- (geeksforgeeks link)

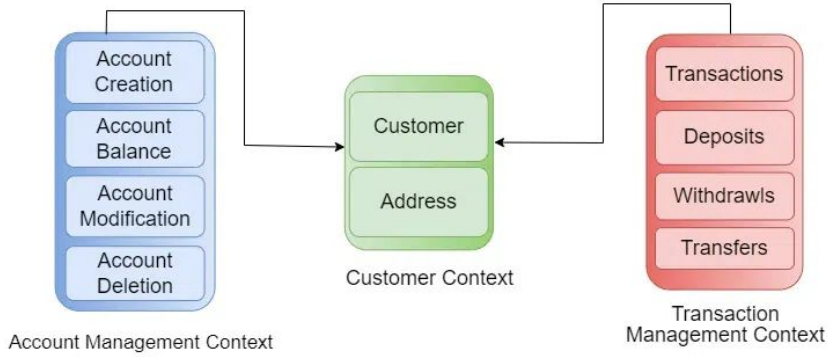
# DDD

What is important to the domain? ➜ **objects/entities**

Identify the objects for the domain
- and their *relationships*
- and their *boundaries*




Context Mapping of Online Banking System

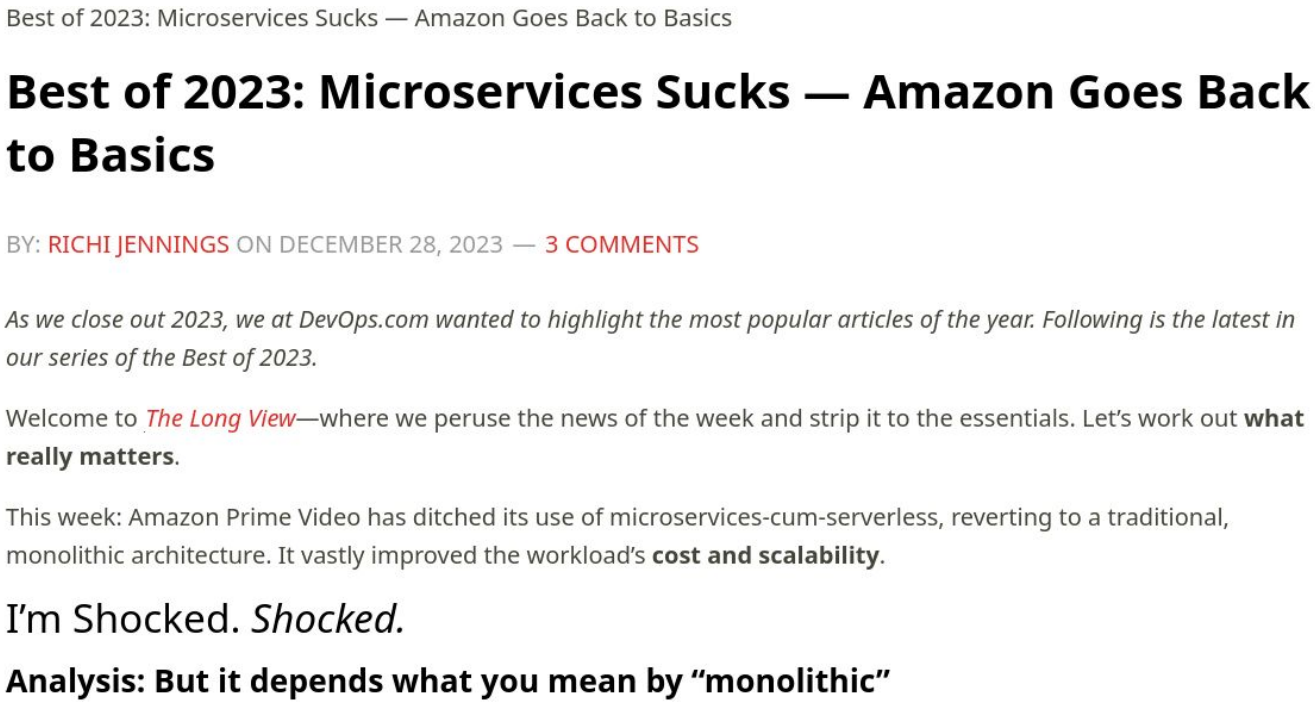



Shared Kernel of Online Banking System

# Is there a good middle ground?

# Oops...was it all a fad?

https://devops.com/microservices-amazon-monolithic-richixbw/



Best of 2023: Microservices Sucks — Amazon Goes Back to Basics

## Best of 2023: Microservices Sucks — Amazon Goes Back to Basics

BY: RICHI JENNINGS ON DECEMBER 28, 2023 — 3 COMMENTS

*As we close out 2023, we at DevOps.com wanted to highlight the most popular articles of the year. Following is the latest in our series of the Best of 2023.*

Welcome to *The Long View*—where we peruse the news of the week and strip it to the essentials. Let's work out **what really matters**.

This week: Amazon Prime Video has ditched its use of microservices-cum-serverless, reverting to a traditional, monolithic architecture. It vastly improved the workload's **cost and scalability**.

### I'm Shocked. *Shocked.*

**Analysis: But it depends what you mean by "monolithic"**

**Not just a scaling advantage?** Rafal Gancarz also notes huge cost savings—"Prime Video Switched from Serverless to EC2 and ECS":

> **"Single application process"**
>
> Prime Video, Amazon's video streaming service ... achieved a 90% reduction in operational costs as a result. ... The initial architecture of the solution was based on microservices ... implemented on top of the serverless infrastructure stack. The microservices included splitting audio/video streams into video frames or decrypted audio buffers as well as detecting various stream defects ... using machine-learning algorithms.
>
> ...
>
> The problem of high operational cost was caused by a high volume of read/writes to the S3 bucket storing intermediate work items ... and a large number of step function state transitions. ... In the end, the team decided to consolidate all of the business logic in a single application process. ... The resulting architecture had the entire ... process running [as] instances distributed across different ECS tasks to avoid hitting vertical scaling limits.

# Read the articles - they're fascinating

However, does this mean that microservices are inherently bad?

# No panacea architecture

Same story, different domain

- Don't over-architect your solution
- Analyze your needs rather than start with pre-conceived notions
  - "We need to use serverless"
- Monoliths have their advantages!
  - The result seems to be a mishmash of monolith with scalable services...

# Is there a good middle ground?

Some combination of microservices swarming around a monolith is probably a decent solution

- Break off the pieces that require frequent updates
- Low-impact to your bill

i.e., design your app for what makes sense for your organization (or your project)

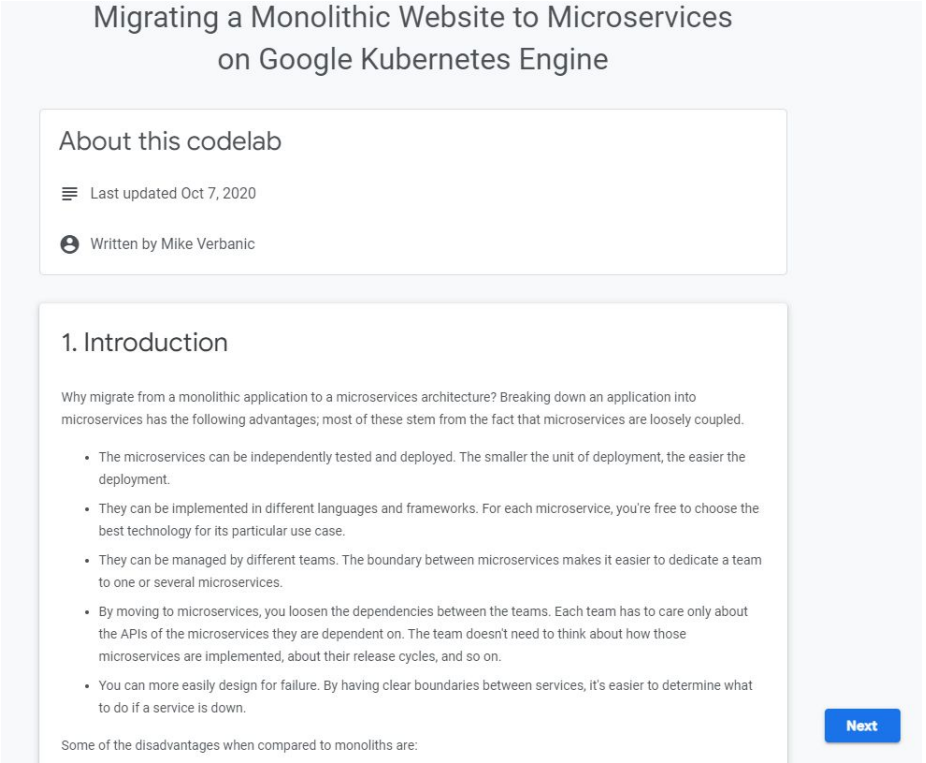
- Don't build to a buzzword!

# Lab time!

Let's go from monolith to microlith (or, collection of microservices)

**CREATE A NEW PROJECT FIRST!**

https://codelabs.developers.google.com/codelabs/cloud-monolith-to-microservices-gke/#0



# What is the goal?

Take an existing monolithic application (defined within Kubernetes) and break it apart into a collection of services

Things to consider:

- There are going to be a lot of commands specific to Kubernetes/Cloud Build here that you don't necessarily need to remember
- Essentially we're going to be:

1) Building and deploying a monolith and checking that it works
2) Taking out part of the monolith and replacing it with another service
    a) And reconfiguring the monolith to point to that service
    b) Building and redeploying and checking that it works
    c) And so on

# If you made a mistake and things are broken

Choices are to:

1) Get frustrated
2) Take a breath
3) See if you made a typo and re-deploy
4) Get frustrated and start over