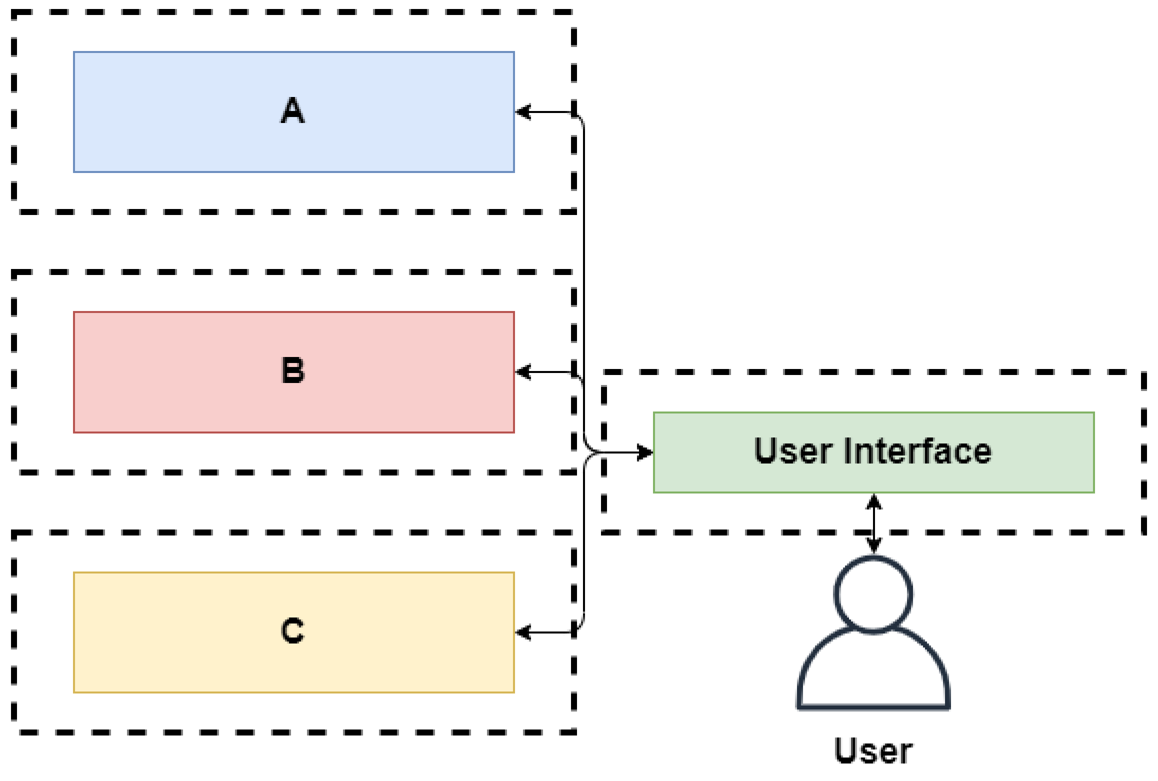
COSC 331 – Microservices and Software

Fall 2020

Recap: What is a Microservice Architecture?

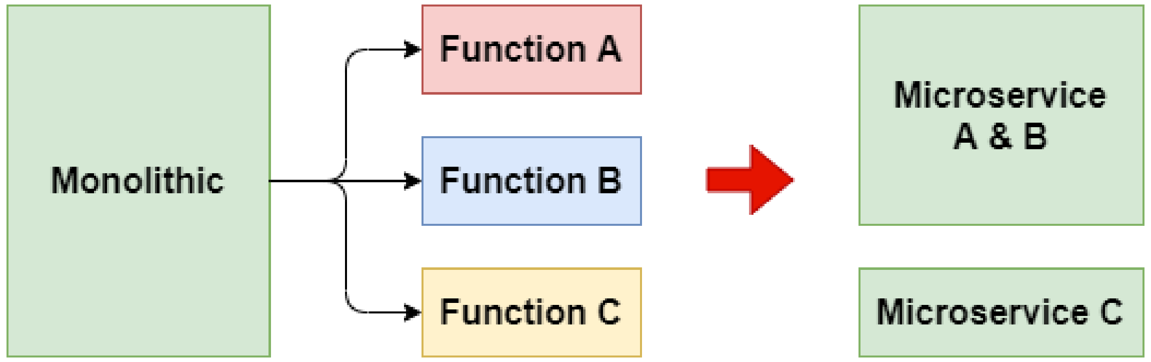
- The microservice architecture replaces a single monolithic service with multiple microservices
- Each microservice represents some small logical component of the larger application



The Scope of a Microservice

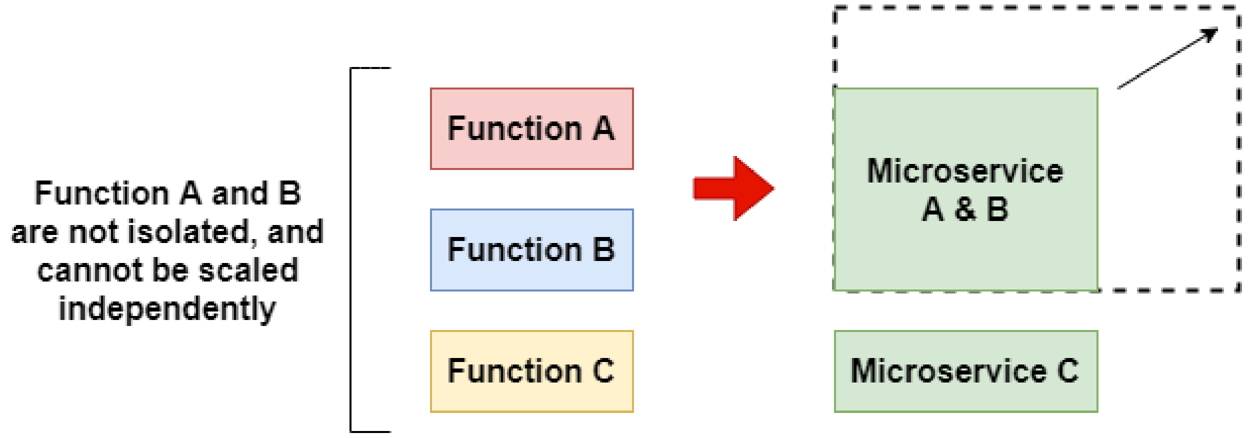
- As mentioned last lecture, a microservice is usually defined by several criteria, chief among them the idea of ***limited scope***
  - A microservice should only implement functionality that is directly related to the narrow application of the microservice
- Core idea here: A microservice should be lightweight and wholly dedicated to doing one task, and doing it well

Why Limit Scope?



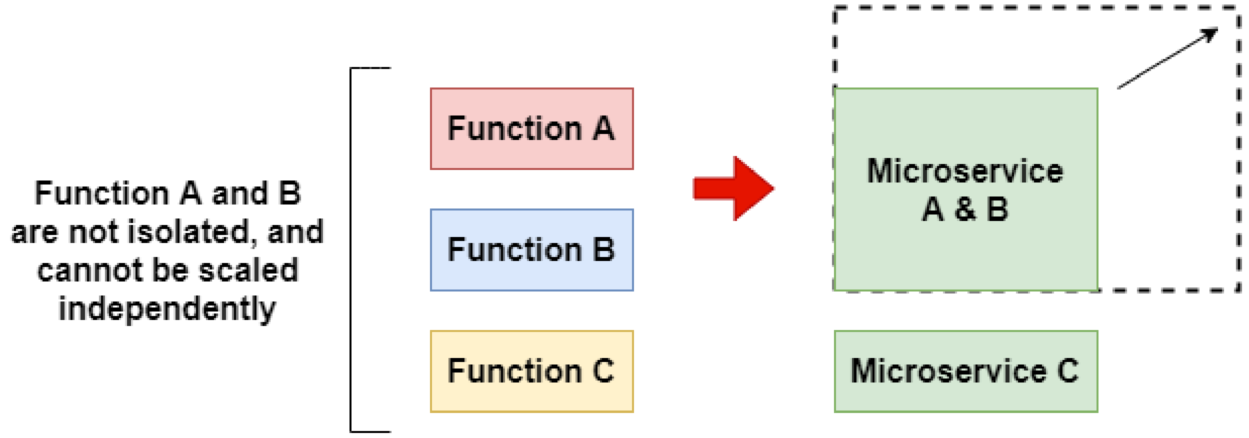
- Consider this example – a monolithic application with three major functions is split into two microservices
- Based on what we heard in the previous lecture, how does this impact ***re-usability***, ***scalability***, and ***isolation***?

Why Limit Scope? Isolation!


Function A and B are not isolated, and cannot be scaled independently
Function A
Function B
Function C
Microservice A & B
Microservice C

- Since function A and B are combined into one microservice, they are no longer isolated from each other
- Failures or faults in function A potentially cause function B to also fail (the service crashes, for example)

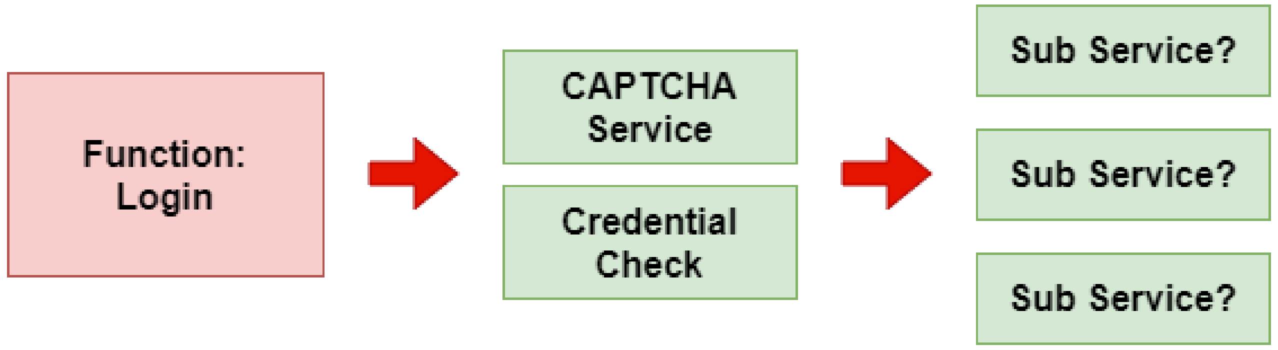
Why Limit Scope? Scalability!


Function A and B are not isolated, and cannot be scaled independently
Function A
Function B
Function C
Microservice A & B
Microservice C

- Similarly, we cannot scale function A and function B separately if they are provided through the same microservice
- If one of them is a bottleneck and the other isn’t, we unnecessarily scale one in order to scale the other (less efficient!)

What is a Realistic Scope?

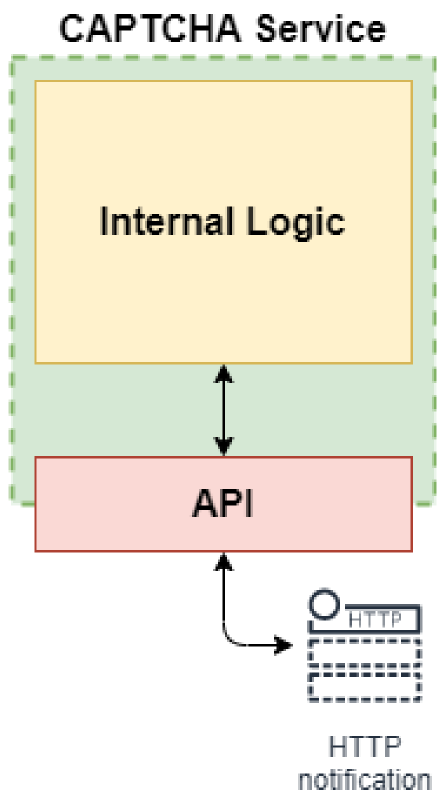
- Of course, it's possible to have ***too many*** microservices!
- We need to draw the line somewhere, otherwise we risk creating dozens of microservices to complete even simple tasks – our application becomes needlessly complex
- Ask yourself: ***is this a reasonable function to isolate?***



The Other Requirements of a Microservice

- Besides being narrowly scoped, our previous criteria for a microservice also had two other components:
  - The microservice should have some sort of **Application Programmable Interface (API)** to handle communication
  - The microservice should use a **common communications protocol** such as HTTP/HTTPS
- What do these criteria imply about our microservice? What kind of system does our criteria need to implement?

The Microservice as Web Application



- If our microservices have an API and communicate via HTTP/HTTPS, they’re essentially just very small web applications
- This means we’ll need to implement some form of web server to listen for and respond to HTTP traffic
- The API acts as a “door”, catching incoming requests and passing that data into the service to be processed

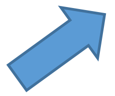
The Application Programming Interface

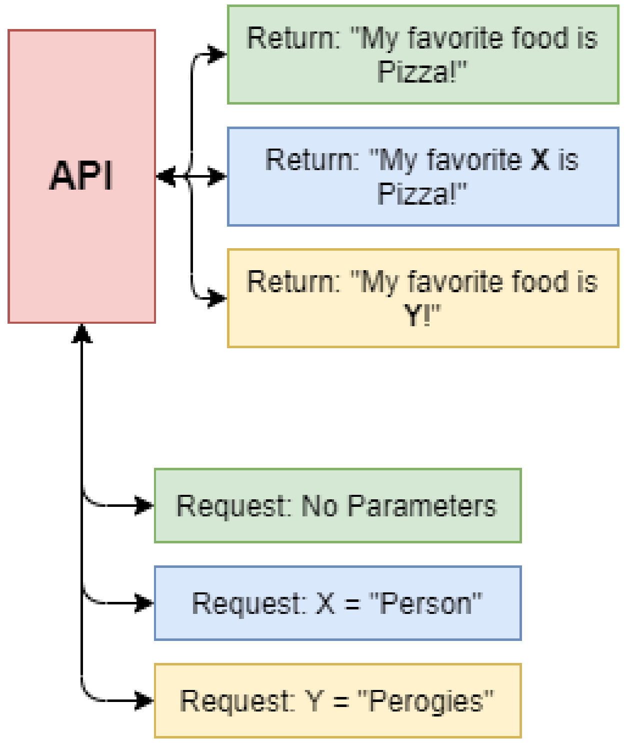
- The API is the main “door” to our microservice – it is the externally visible part of our service
- An API can be as simple as a single input (a GET request) which produces a single HTTP response – the service takes only one kind of request, and returns only one kind of response
- In a more complex service, the API might accept many possible inputs (different parameters, different types of request) and produce multiple possible outputs

The API in Action

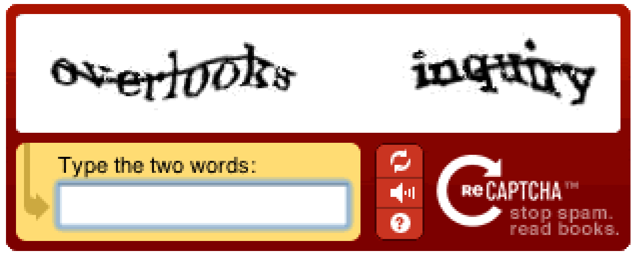
- APIs are designed to provide an organized and documented way to interface with an application
- Often implemented using HTTP GET requests, using parameters stored within a **query string**
- Using the example, an API endpoint might be:

  www.myservice.com?**Y=Perogies**




API
Return: "My favorite food is Pizza!"
Return: "My favorite **X** is Pizza!"
Return: "My favorite food is **Y**!"
Request: No Parameters
Request: X = "Person"
Request: Y = "Perogies"

A Real-World Example

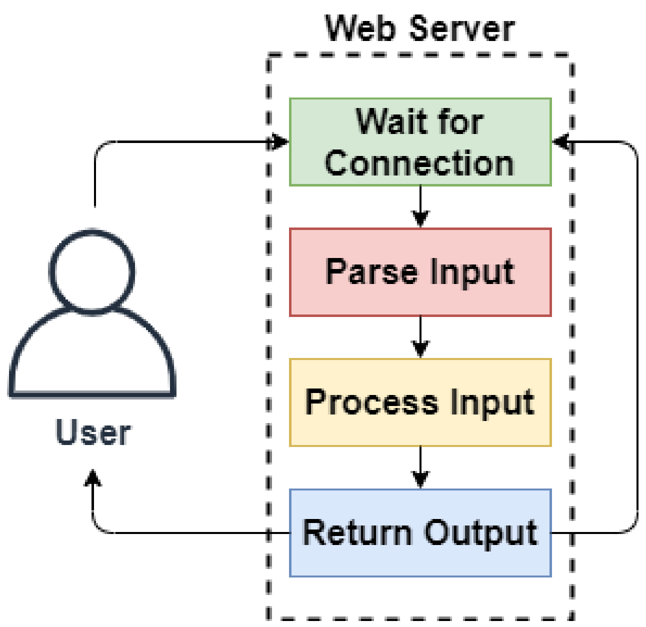


- Consider the CAPTCHA service – a system designed to check if the user is a human, and allow only humans to pass
- First question: Would it be worthwhile to split CAPTCHA into multiple sub services?
- Second question: What are the inputs and outputs of the CAPTCHA service?
- Third question: What is the CAPTCHA service actually communicating with? The client, or the application? Does it matter? (hint: it definitely does)

Implementing the Web Server

- You may have experience working with web server software in previous courses, such as Apache
- However, we want our microservice to be lightweight, and we probably don’t need all the functionality that Apache comes with (designed for serving full web pages)
- So what do we do? We’ll implement our own web server to handle the traffic!
- We’ll do this using **sockets**

A Simple Web Server



- A simple web server opens a specific ***socket***, which can be reached at a specified ***port***
- It waits for a user to connect and then parses any input the user has submitted (parameters, input data, etc)
- It then processes this data and produces an output, which it returns to the user
- The server then goes back to waiting for another connection – loop continues for as long as the service is running

Lab One – Developing a Simple Microservice

- For our first lab, we'll be designing a simple microservice using ***Java***
- Our microservice will return a special customized Cascading Style Sheet (CSS) to add custom styling to a web page
- The client will be able to pass certain parameters (color, font) to our microservice, which will use these parameters to generate a basic CSS file with the requisite colors and font choices
- This sheet will then be returned to the user and used to style a web page to the user's preferences

Lab One – What Inputs?

- Lets consider what inputs our CSS microservice will require:
  - Main color: the main background color of the page
  - Accent color: a secondary color for accents
  - Font: the font to use for the page
- First, what do you think the best way to send this data is? What is a compact and straightforward way to send color data?

Lab One – The Processing Aspect

- Once we've received our color and font inputs from a client, we should then process this input
- First, we'll need to ***parse*** the input – extracting the useful parameters from the incoming HTTP request and passing it off to our processor
- Then, the software will need to substitute in our parameters for values within an already-defined CSS file (i.e. performing a string replacement)

Lab One – The Output Aspect

- We need to think a little bit about how we’re returning our output – what do we need to consider?
- Since we want to use our output CSS as a style sheet for a web page, we need to make sure our output is an actual CSS file so the web browser will know how to handle it
- Once we’ve returned our output, we’ll want our service to return to it’s waiting phase until another request is made

But I’ve never worked with Sockets!

- For our first lab, we’ll be using ***Java Sockets*** to handle our incoming/outgoing HTTP traffic
- Sockets provide a low-level network interface, giving us a great deal of control over how we handle input and output
- The downside is it’s more effort, as we’ll need to implement parsing and connection control ourselves
- But don’t worry – the lab will include sample code and a tutorial on how to use socket connections in Java
  - Not actually as hard as you would think

Prepping for the Lab

- Before the first lab, you should download an appropriate Java IDE (Integrated Development Environment) – use whatever you’re most comfortable with
- If you’re not sure which IDE to use, Eclipse is free and probably the most common Java IDE currently in use – plenty of documentation is available for it if you’re having difficulties
- Run a simple “Hello World” program to ensure that you have all the requisite Java runtime dependencies installed and correctly configured

Any Questions?