Microservices

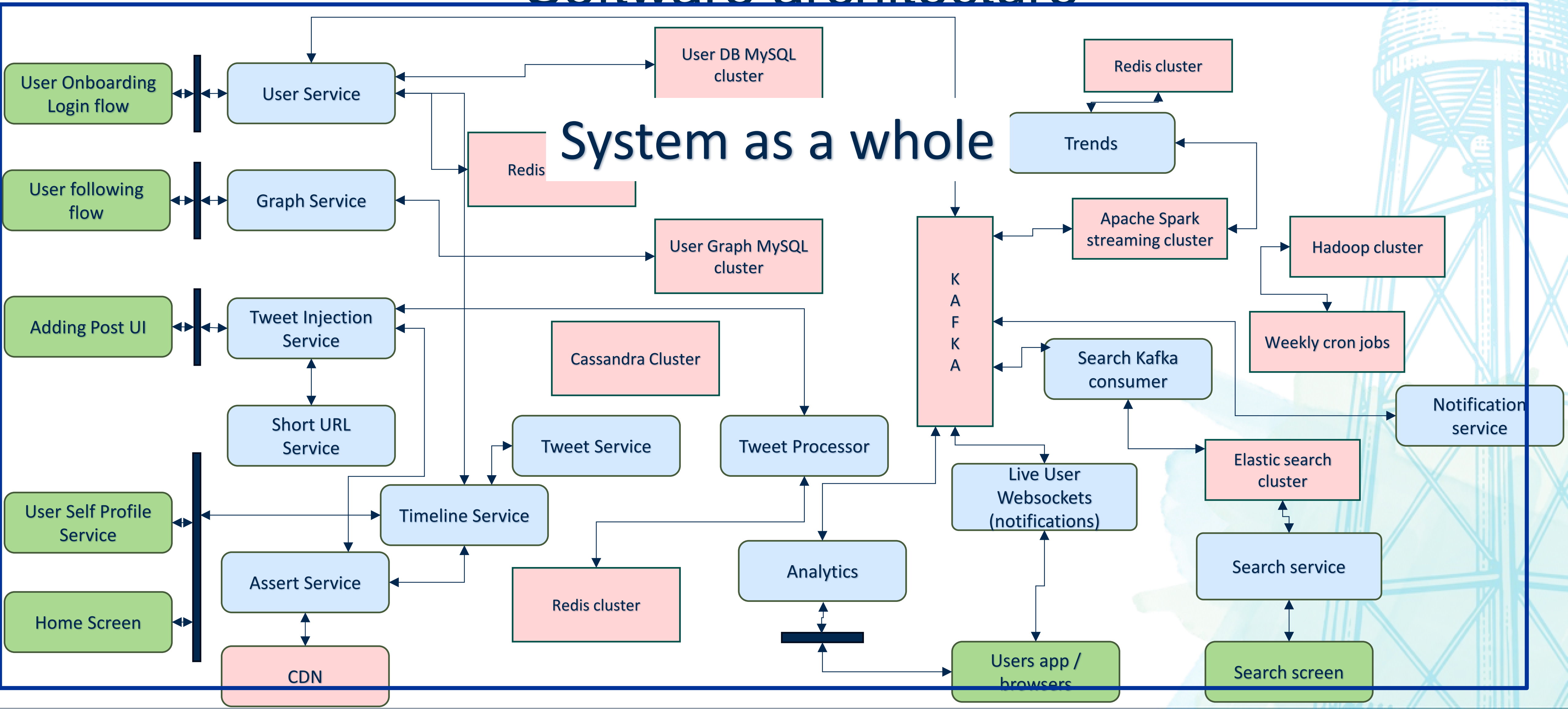
Tapti Palit

UCDAVIS

Outline

- Service boundaries
- Communication (sync vs. async)
- Data per service
  - Logs (events)
  - LSM (state)
  - In-memory cache

Software architecture



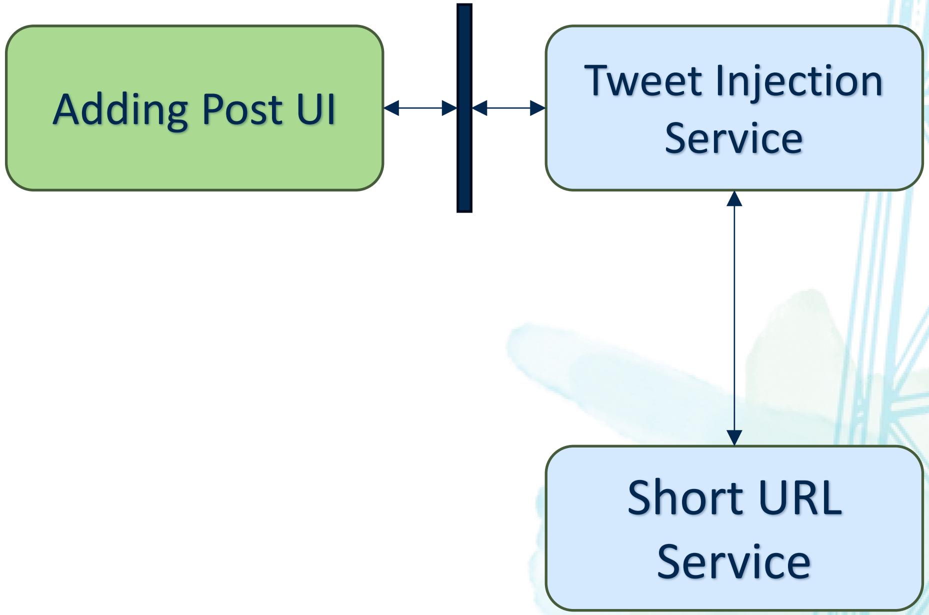
UCDAVIS

Some software architecture concerns

- Encapsulation
- Scalability
- Resilience
- Performance
- Ease of deployability
- Security (ECS 153)

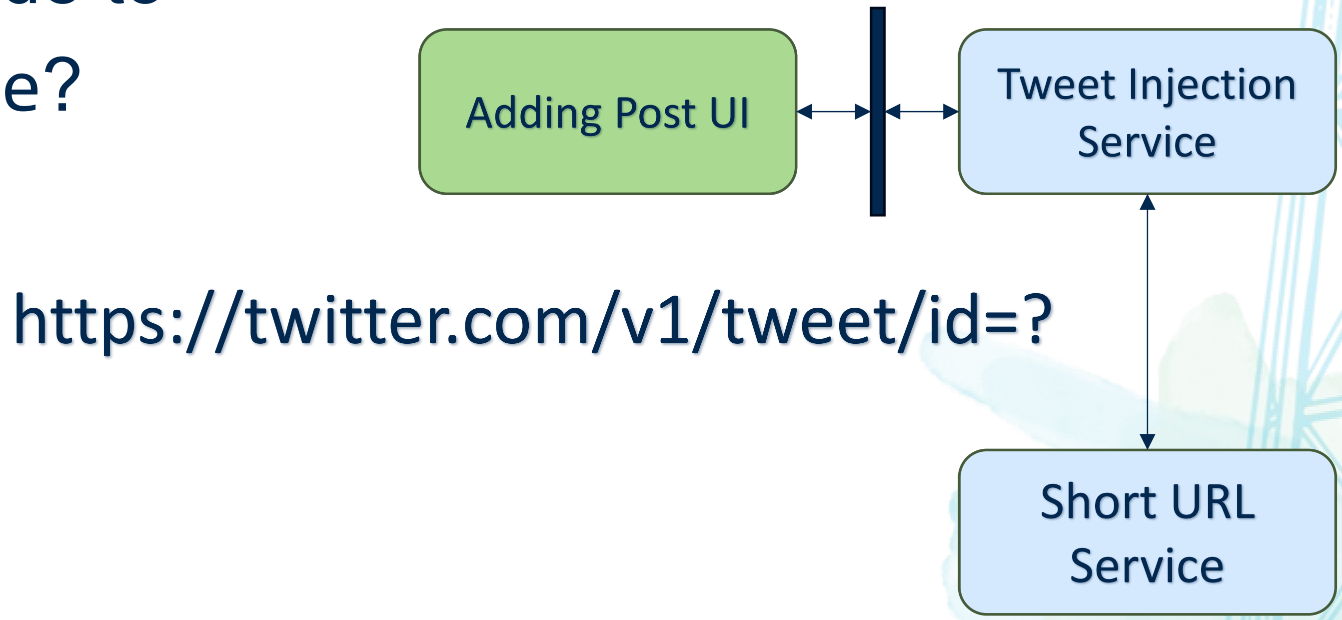
Encapsulation

- At the component level
  - Does `TweetInjectionService` need to know anything about the ***internals*** of `ShortURLService`?



Encapsulation

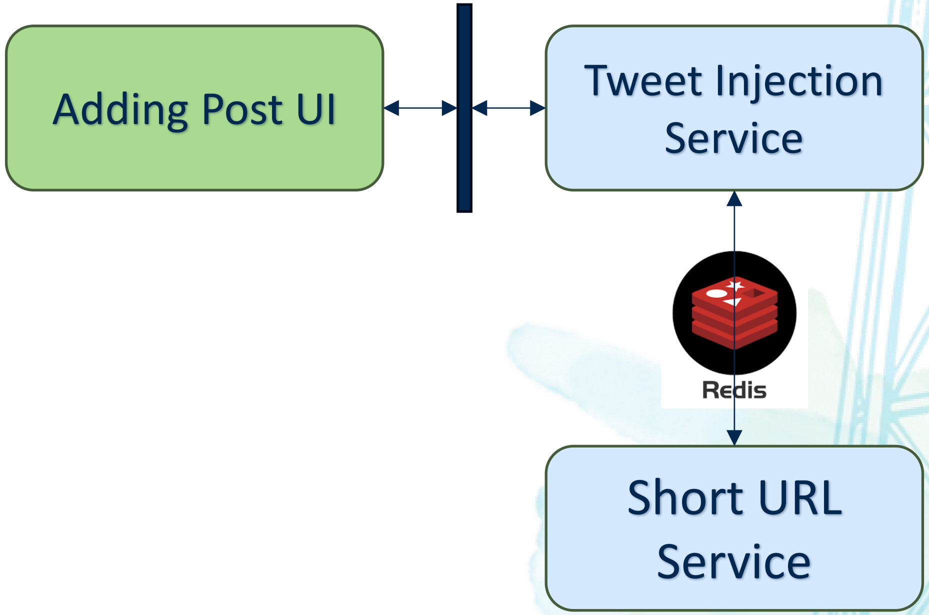
- What interface does `ShortURLService` provide to `TweetInjectionService?`



UCDAVIS

Encapsulation

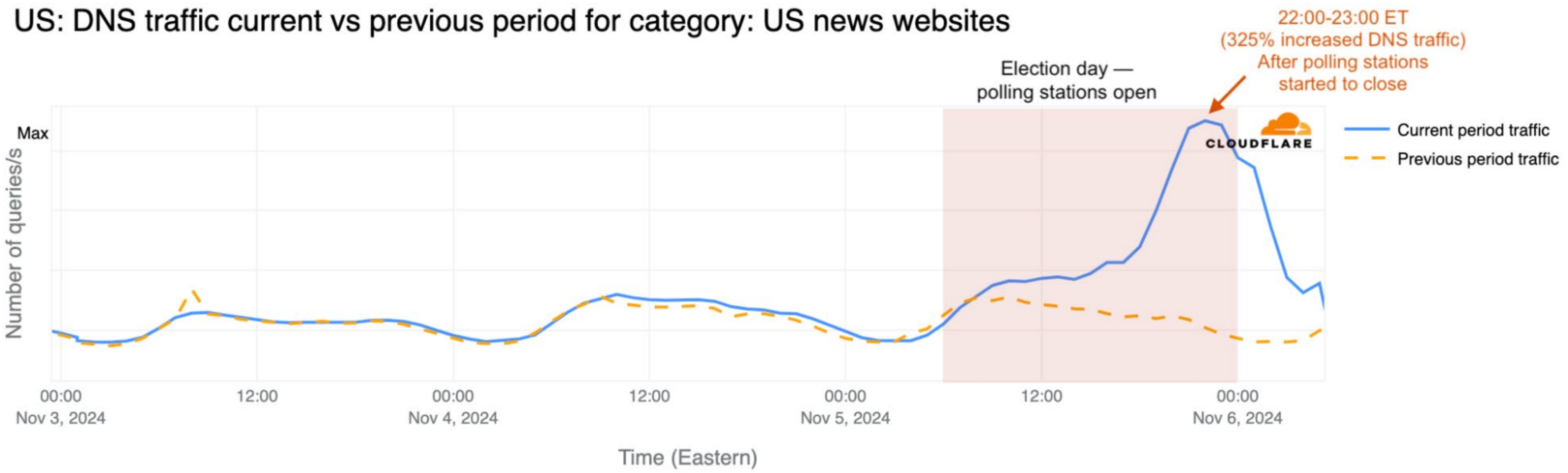
- Typically run on different machines
  - How do they ***communicate***?
  - Directly call each other or is there a mediator?
    - Different tradeoffs



Scalability

- Website traffic is not constant
- Can spike due to planned events
  - Product launch
  - US elections

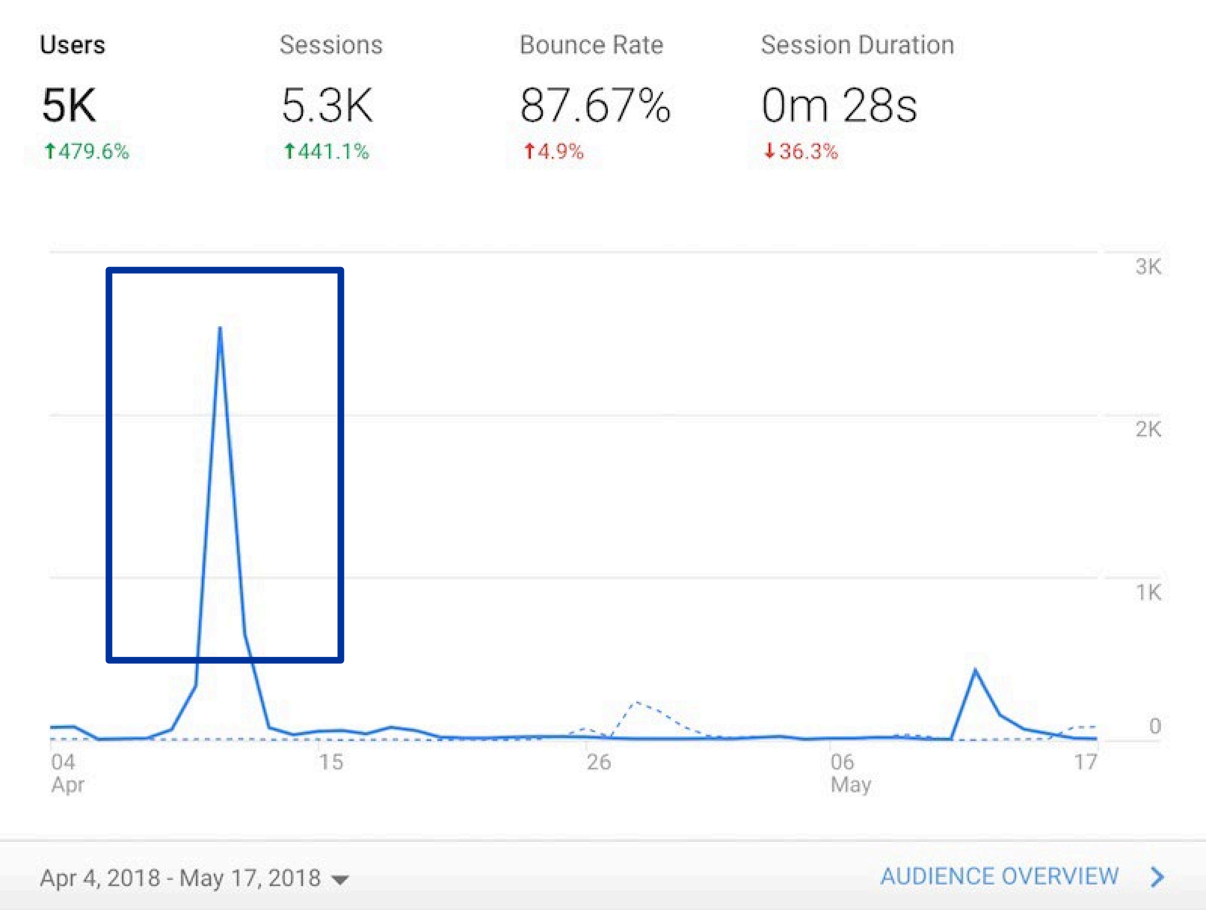
https://blog.cloudflare.com/exploring-internet-traffic-shifts-and-cyber-attacks-during-the-2024-us-election/



Scalability

- Unplanned events
  - Post goes viral
- Architecture should *scale* to handle such events

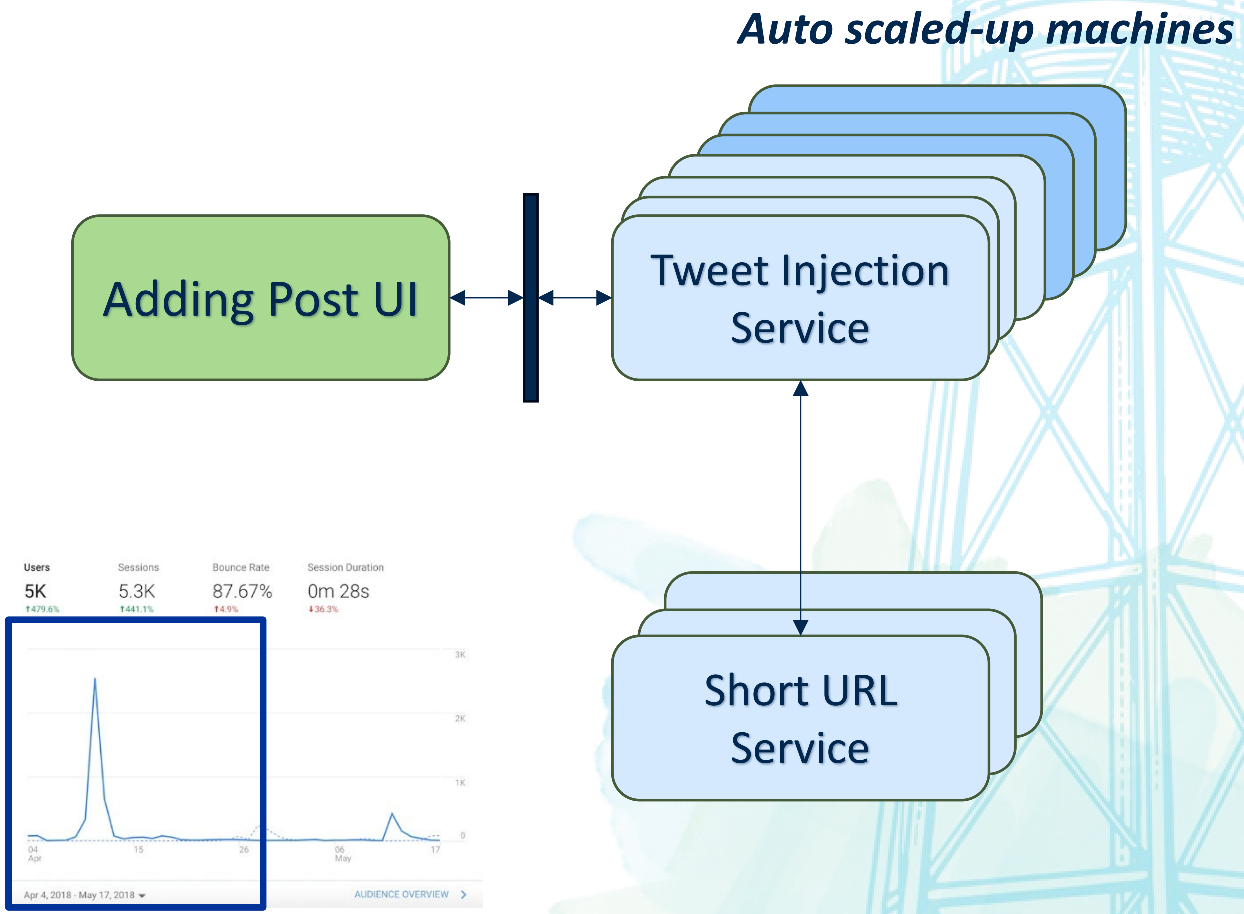
https://www.residualthoughts.com/2018/05/20/traffic-data-from-a-viral-post/



UCDAVIS

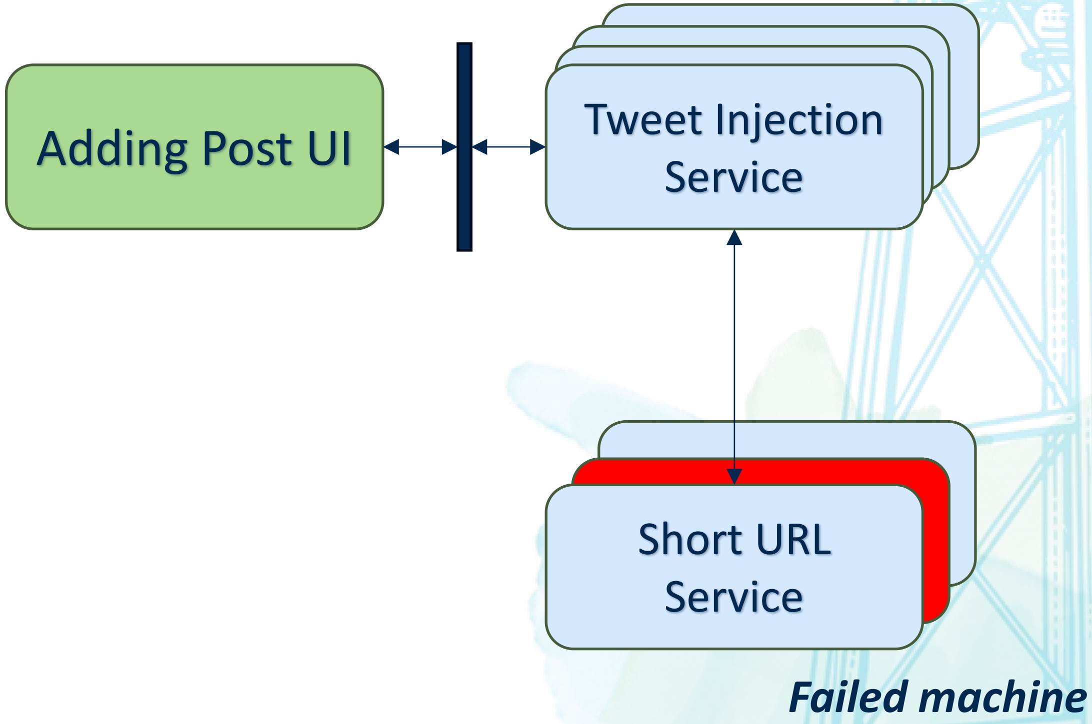
Scalability

- Services run on more than one machine
- Autoscaling for when traffic reaches threshold
  - System automatically brings up new “machines”
  - Available in popular cloud deployments
    - Amazon AWS, Digital Ocean, Google Cloud



Resilience

- Is the system resilient to failures?
  - If a single component fails does the entire system fail?

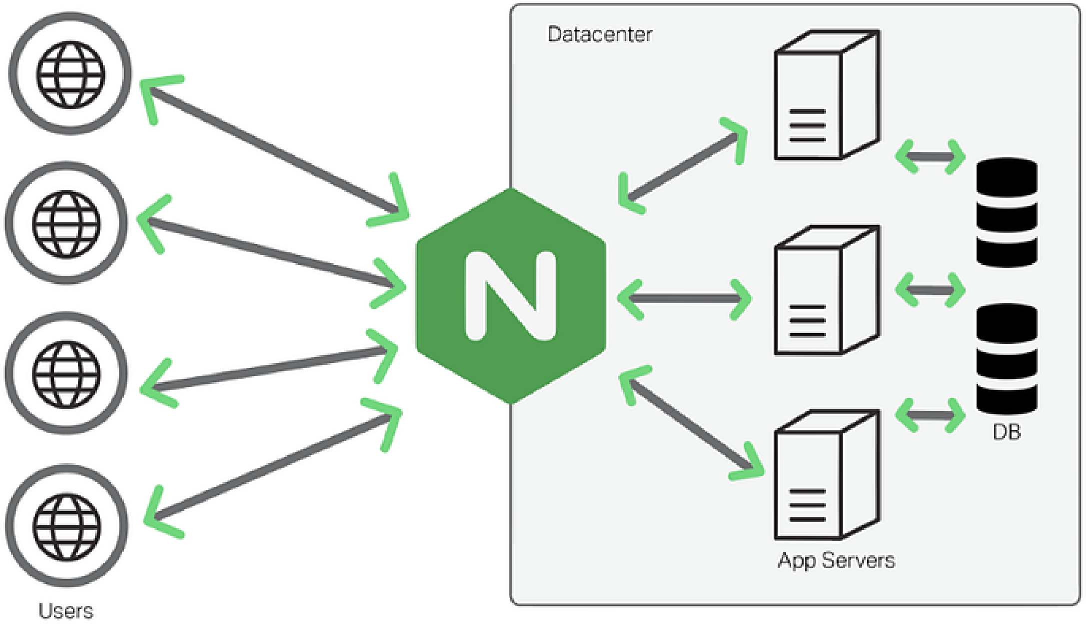


Resilience

- Fault tolerance
  - Distribute traffic across multiple instances (load balancers)
  - Load balancer detects the failures and stops sending requests to faulting nodes

HAPROXY

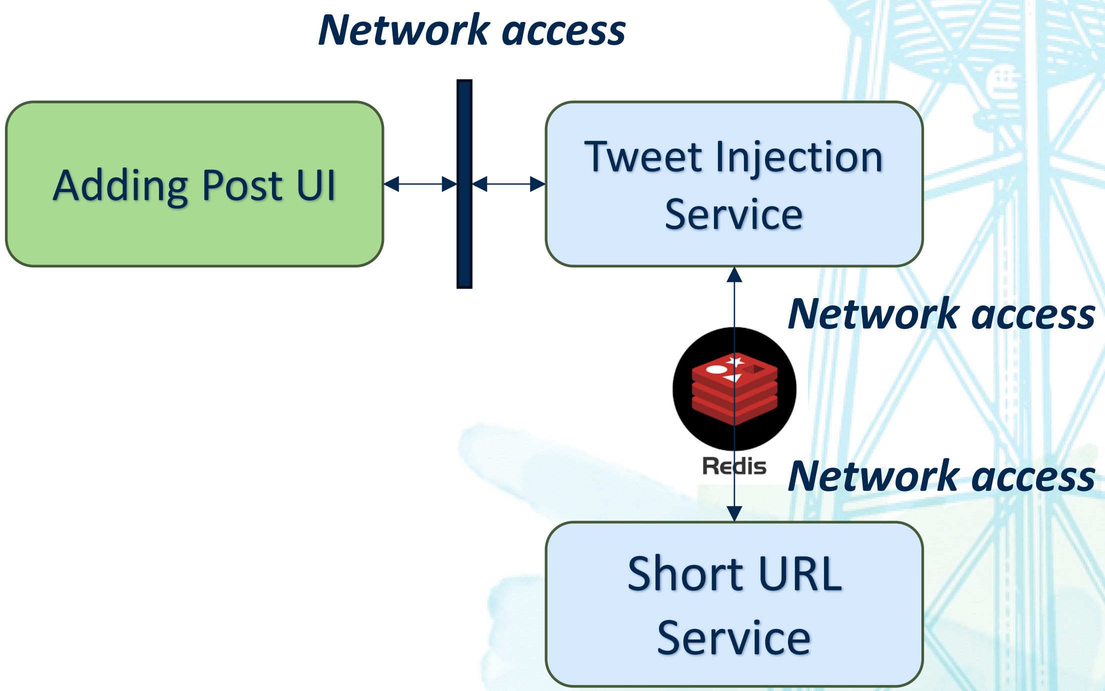
NGINX



Nginx load balancer

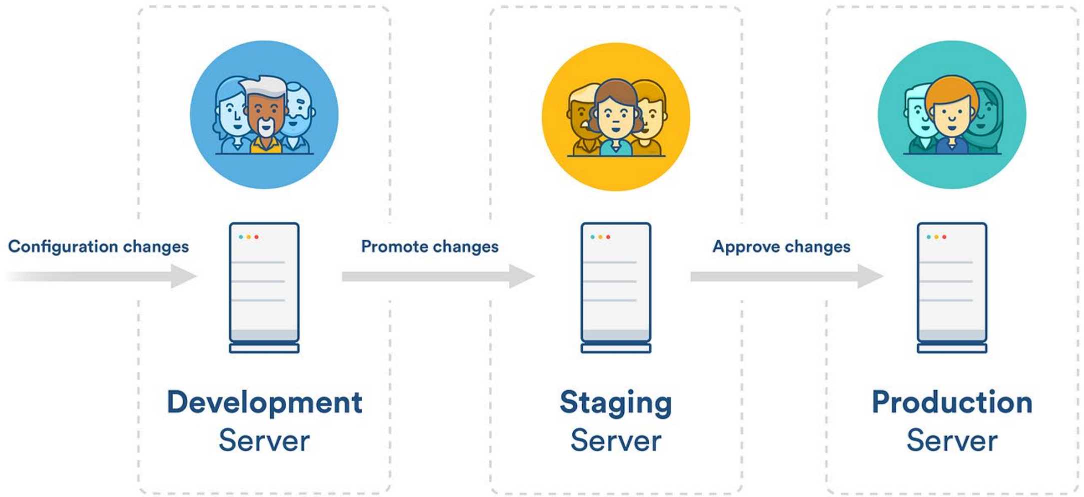
Performance

- Does the architecture itself cause any performance bottlenecks?
- Modularity reduces performance
- How to limit performance overhead?



Deployability

- Can individual components be individually deployed?
  - Do I need to deploy TweetService to deploy ShortURLService?
- Can deployment steps be automated?



Dimensions of software architecture styles

- System organization
- Data organization
- Communication architecture

Dimensions of software architecture styles

- System organization
  - Monolithic and microservices
- Data organization
- Communication architecture

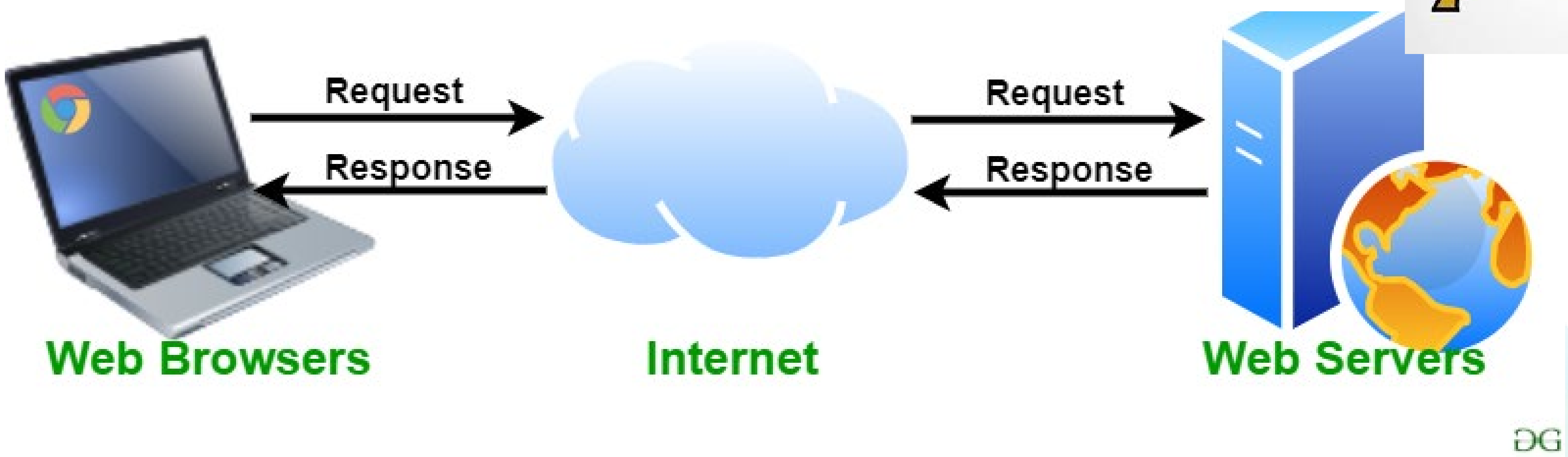
Dimensions of software architecture styles

- System organization
  - Monolithic and microservices
- Data **per service**
  - LSM (state)
  - Logs (events)
  - In-memory caches
- Communication architecture

Dimensions of software architecture styles

- System organization
  - Monolithic and microservices
- Data **per service**
  - LSM (state)
  - Logs (events)
  - In-memory caches
- Communication architecture **for microservices**
  - Synchronous (RPC)
  - Asynchronous (Message queuing, pub-sub)
  - Event-driven (Kafka – next module)

Client server architecture



Pros and cons

- Benefits
  - Centralized management of data: all data is on the server and can be easily secured
    - No complex data flows
  - Separation between client and server business logic
- Disadvantages
  - Single point of failure
  - Poor scaling

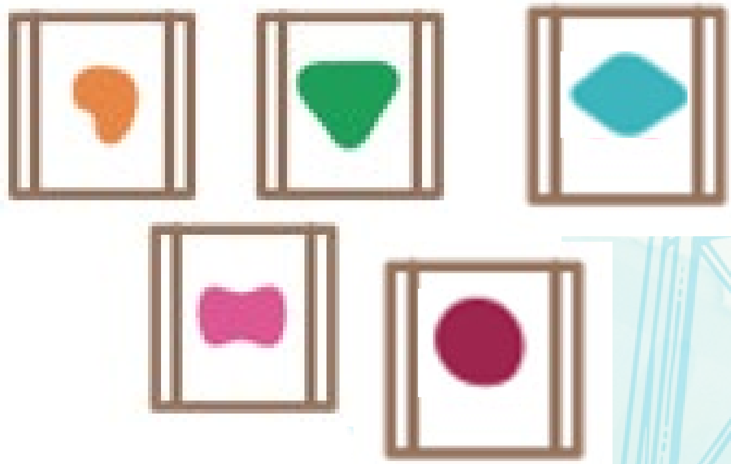
Microservices

- An approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API https://martinfowler.com/articles/microservices.html
- Independently deployable by automated processes
- Bare minimum centralized management
- Smart endpoints connected by “dumb” pipes

Microservices overview

- An architectural style for building distributed systems
- Applications are divided into small, independent services

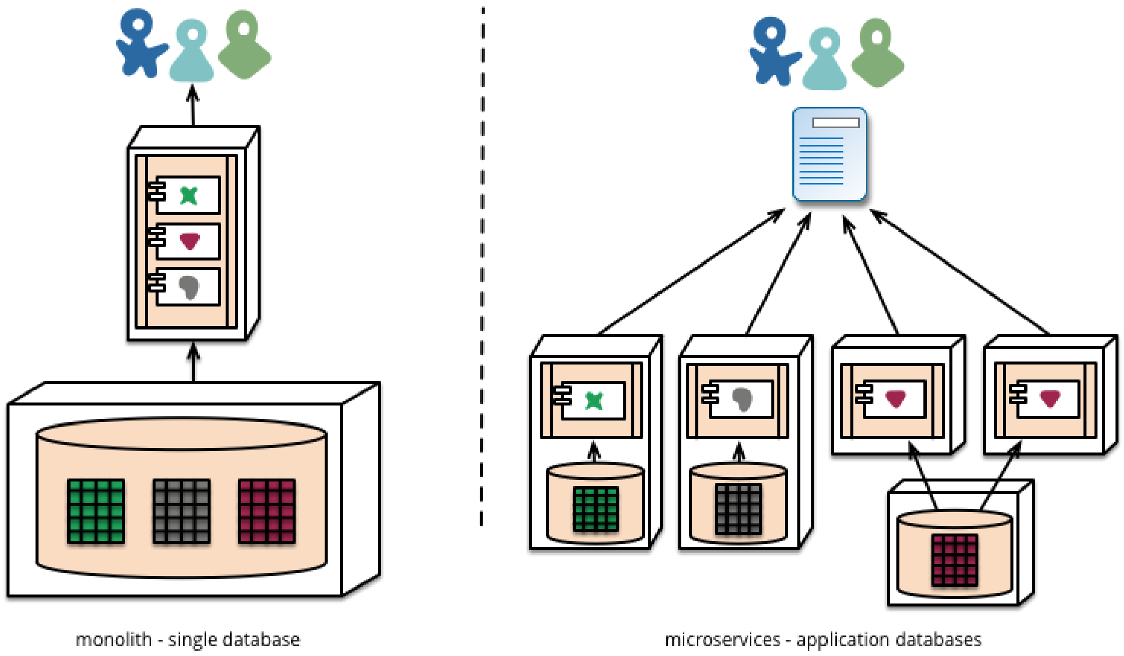
Monolithic application
(Multiple components running in same process on same machine)



Microservices
(Each component runs in its own process on potentially its own machine)

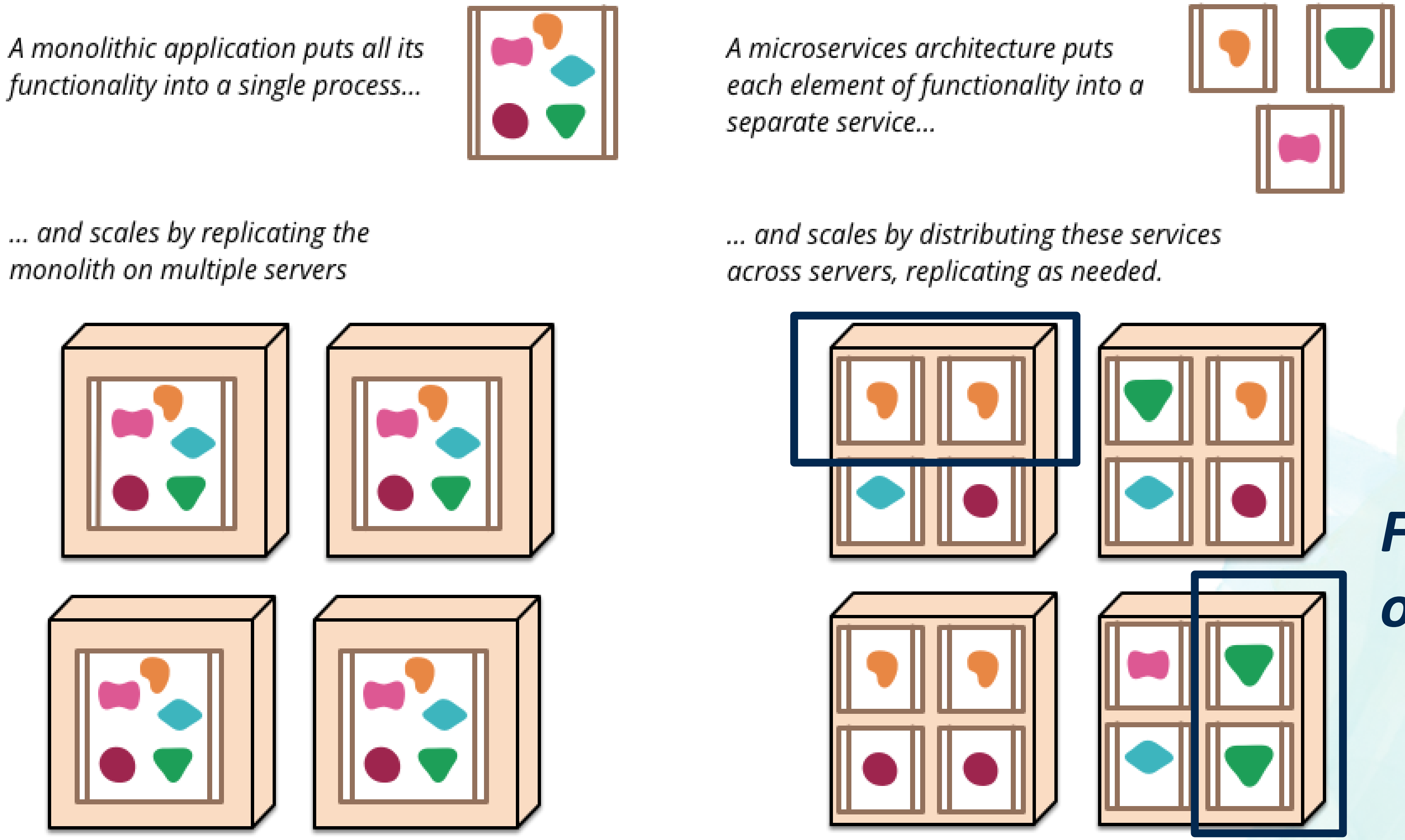
Monolithic to microservices

- Each microservice can have its own database – ***shared nothing architecture***



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

Microservices provide finer scaling control



Finer control over scaling

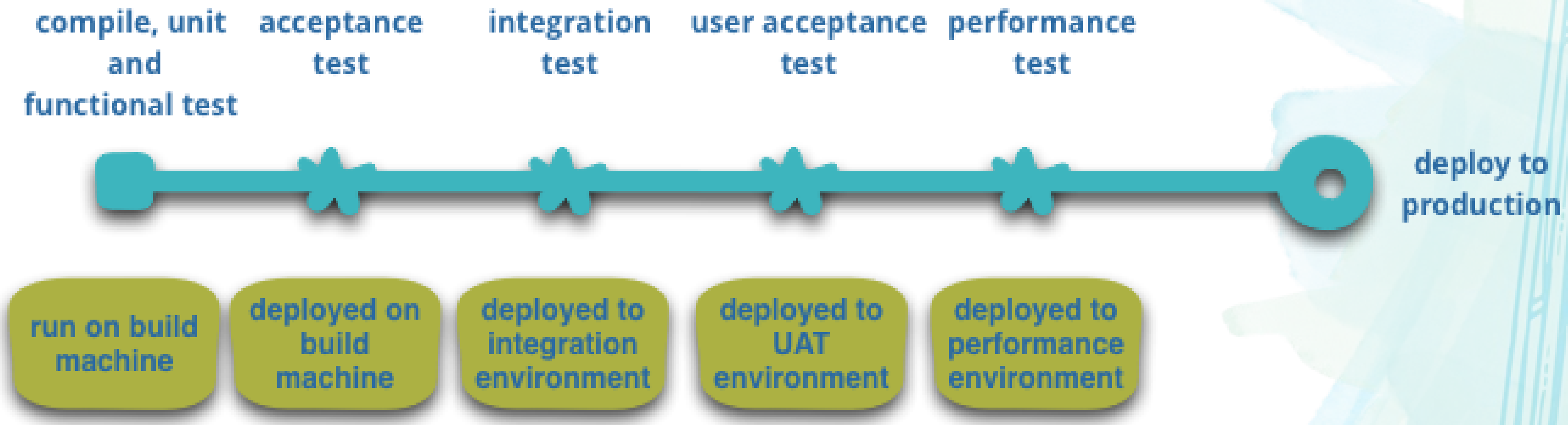
https://martinfowler.com/articles/microservices.html

Microservices provide finer scaling control

- Monolithic application can only be deployed as a whole
  - Therefore, can only be scaled as a whole
- Microservices are individually deployable
  - Therefore, they provide finer scaling control

Continuous deployment using microservices

- Microservices simplify automated deployment
- E.g., pushing to the main branch in GitHub triggers GitHub Actions that
  - Automatically builds
  - Execute unit-tests
  - Automatically deploy to integration environment (and then production environment)



Pros and cons

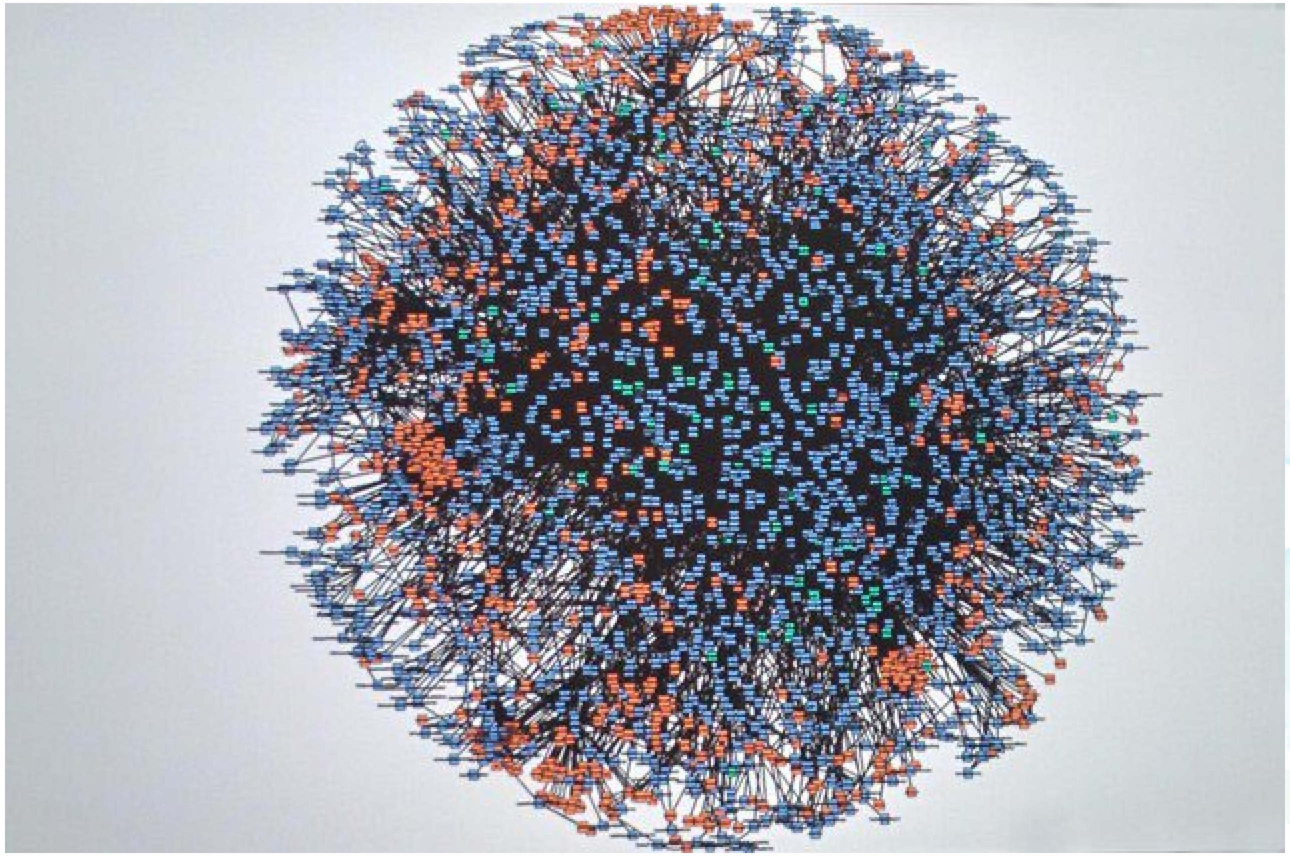
- Pros
  - Strong encapsulation and modularity
  - Better reusability
  - Each microservice can be scaled independently *(More on this later)*
  - Each microservice can be written in its own programming language
  - Fault isolation
  - Supports CI/CD (easier to deploy microservices than monolithic services)
- Cons
  - Higher complexity
  - Debugging complex interactions is harder
  - Network overhead

Microservices at Uber (2019)

https://x.com/msuriar/status/1110244877424578560

Microservices at Amazon (2008)

- Code-named “Deathstar”



https://x.com/Werner/status/741673514567143424

Spring Boot Overview

- Framework for creating RESTful microservices
- Reduces boilerplate configuration code
- Embedded server (Tomcat/Jetty)
- Simplifies microservice creation through annotations
- Built-in support for REST APIs

RESTful microservices with Spring Boot

- Create classes that can act as REST endpoints
- Uses annotations to denote REST endpoint URLs
  - Allows complete decoupling from the boilerplate code
- Types of requests
  - `@GetMapping`, `@PostMapping`, `@PutMapping`, and so on… for all HTTP methods
- `@PathVariable` – extract variable from GET request
- @RequestBody – extract the post request body

```
class MyRequest {
    private String postDate;
    private String postContent;
    // .. Getters and setters
}

@RestController
@RequestMapping("/myservice")
public class MyController {
    @PostMapping("/sayhello")
    public String sayHello(@RequestBody MyRequest
request) {
        return "";
    }
}
```

Effective URL: http://[serverip]/myservice/sayhello

Spring Boot Framework

- Uses reflection to first look up all classes with @RestController annotation
- Then automatically creates Servlets out of the methods annotated with @GetMapping, @PostMapping, etc.
- Uses reflection to parse the request parameters into class objects annotated with @RequestBody
- Generates the WAR file and launches the Apache Tomcat server
  - Simply execute `mvn spring-boot:run`

```
class MyRequest {
    private String postDate;
    private String postContent;
    // .. Getters and setters
}

@RestController
@RequestMapping("/myservice")
public class MyController {
    @PostMapping("/sayhello")
    public String sayHello(@RequestBody MyRequest
request) {
        return "";
    }
}
```

Spring Boot demo

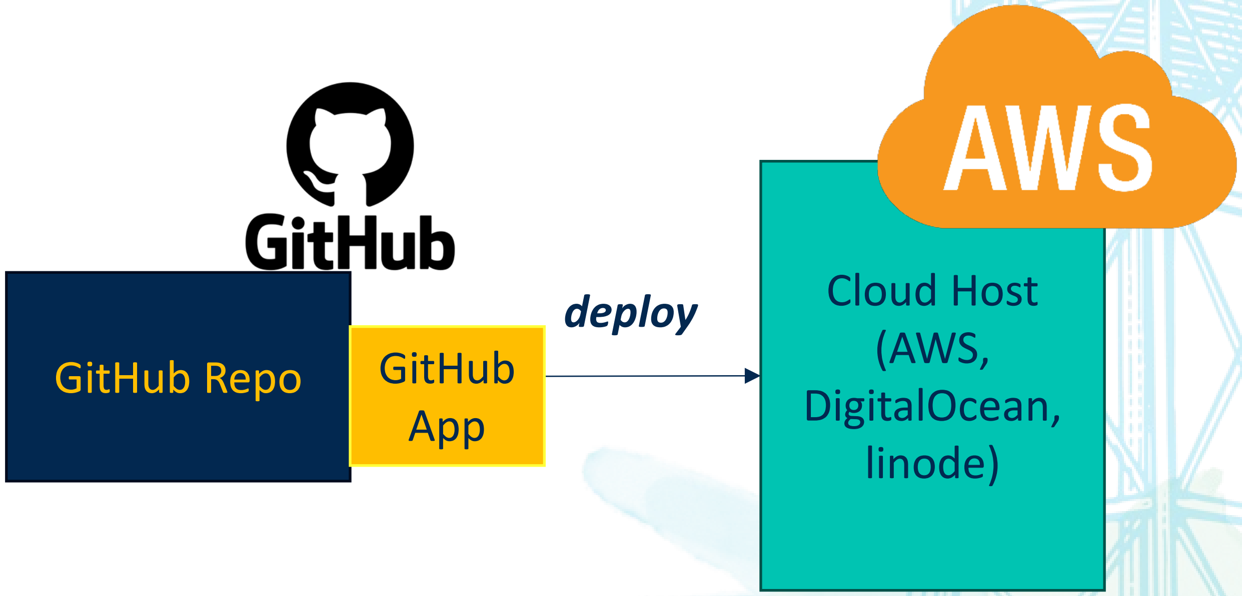
- https://www.youtube.com/watch?v=uGDhkWc4gYA

Continuous Deployment

- Automated release of every code change that passes tests directly to production
- Final stage of a modern CI/CD pipeline
- Uses containers
- GitHub Actions OR GitHub App Integration through cloud host
- GitHub Apps can ***extend the functionality*** of GitHub itself

Cloud host integration

- Every push to the repo must automatically deploy to the cloud
- How to deploy?
  - Provide a Dockerfile that packages and launches the software
- How to detect new commit pushes?
  - GitHub Actions
  - GitHub Apps provided by the Cloud Host



Demo – DigitalOcean integration

- DigitalOcean is a cloud provider, like AWS
  - Can launch VMs
- Apps Platform
  - Managed service that automatically builds, deploys, and scales your apps
  - Directly from GitHub
- Following steps similar across many platforms

Containerize your app

- Containers build on a base image
  - maven:3.9.8-eclipse-temurin-21
    - https://hub.docker.com/layers/library/maven/3.9.8-eclipse-temurin-21/images/sha256-d86bfd73fbfd7e9f0a9554ae23689bd46289f537df4a5aa368fdd1a8c25dd8d0?context=explore
  - eclipse-temurin:21-jre

```
# --- Build stage ---
FROM maven:3.9.8-eclipse-temurin-21 AS build
WORKDIR /app
COPY pom.xml . # Copy pom.xml into the container
COPY src ./src
RUN mvn -q -DskipTests package # Build it!

# --- Runtime stage ---
FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
CMD ["java", "-jar", "app.jar"]
```

Message queues and pub-sub architecture

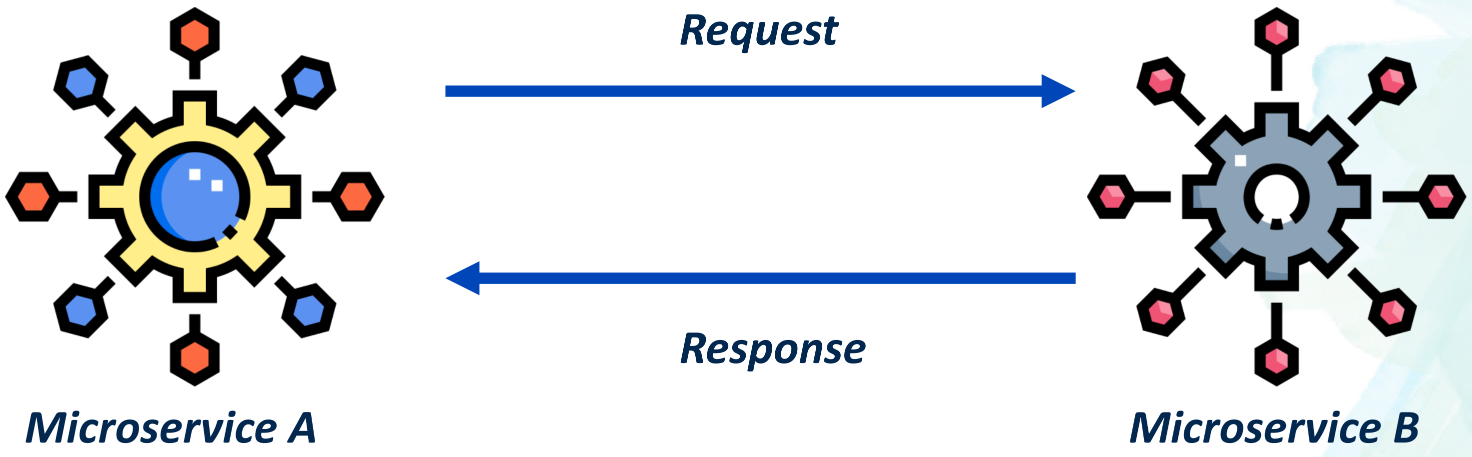
Tapti Palit

Outline

- RPC
- Message queue
  - Data-structures
  - Persistence
- Pub/sub
  - Data-structures
  - Persistence

Remote procedure call (RPC)

- Library/framework gives the illusion that the other microservice is running locally
  - Protocol that allows a program to execute a procedure on a remote server as if it were local
- Synchronous communication – caller waits for response before proceeding



RPC key features

- Language agnostic: the RPC itself does not depend on the service language
- Abstracts network details
  - Typically, over HTTP

RPC formats

- Text-based (e.g. REST APIs)
  - Uses JSON or XML for data exchange
  - Human-readable, but larger payloads
- Binary formats (e.g. gRPC)
  - Uses binary standards (such as `protobuf`) for serialization
  - Compact but faster, but less human-readable
    - … ***why?***

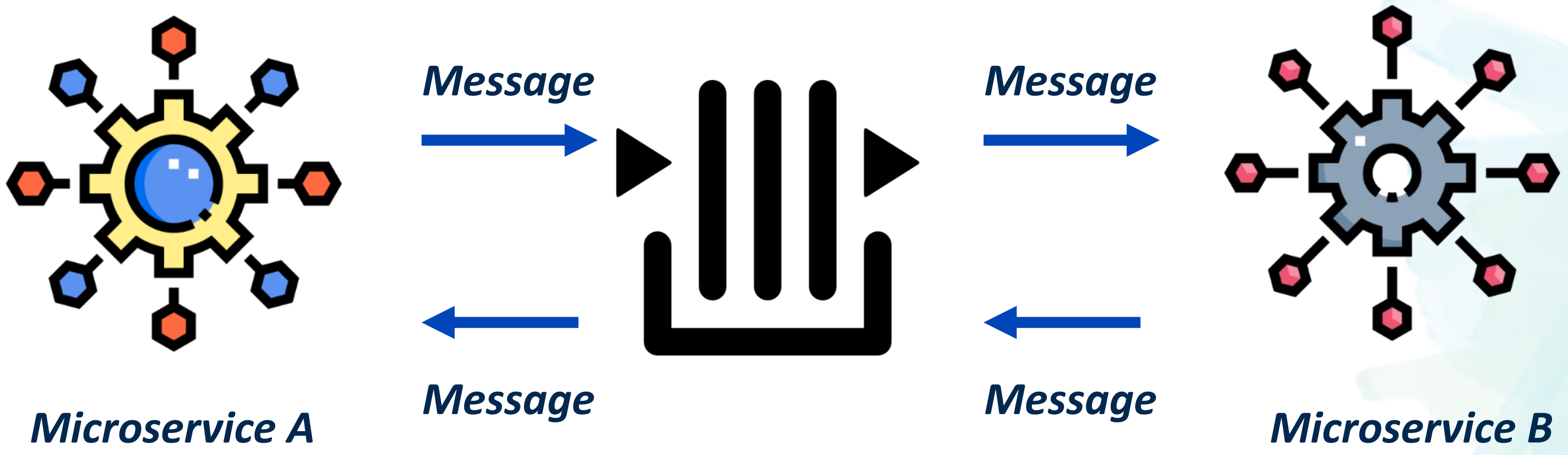
JSON and REST APIs

- Representational State Transfer (REST) is an architectural style for web services
- Application/microservice exposes an URL
- Uses HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources
- Commonly paired with JSON for data exchange

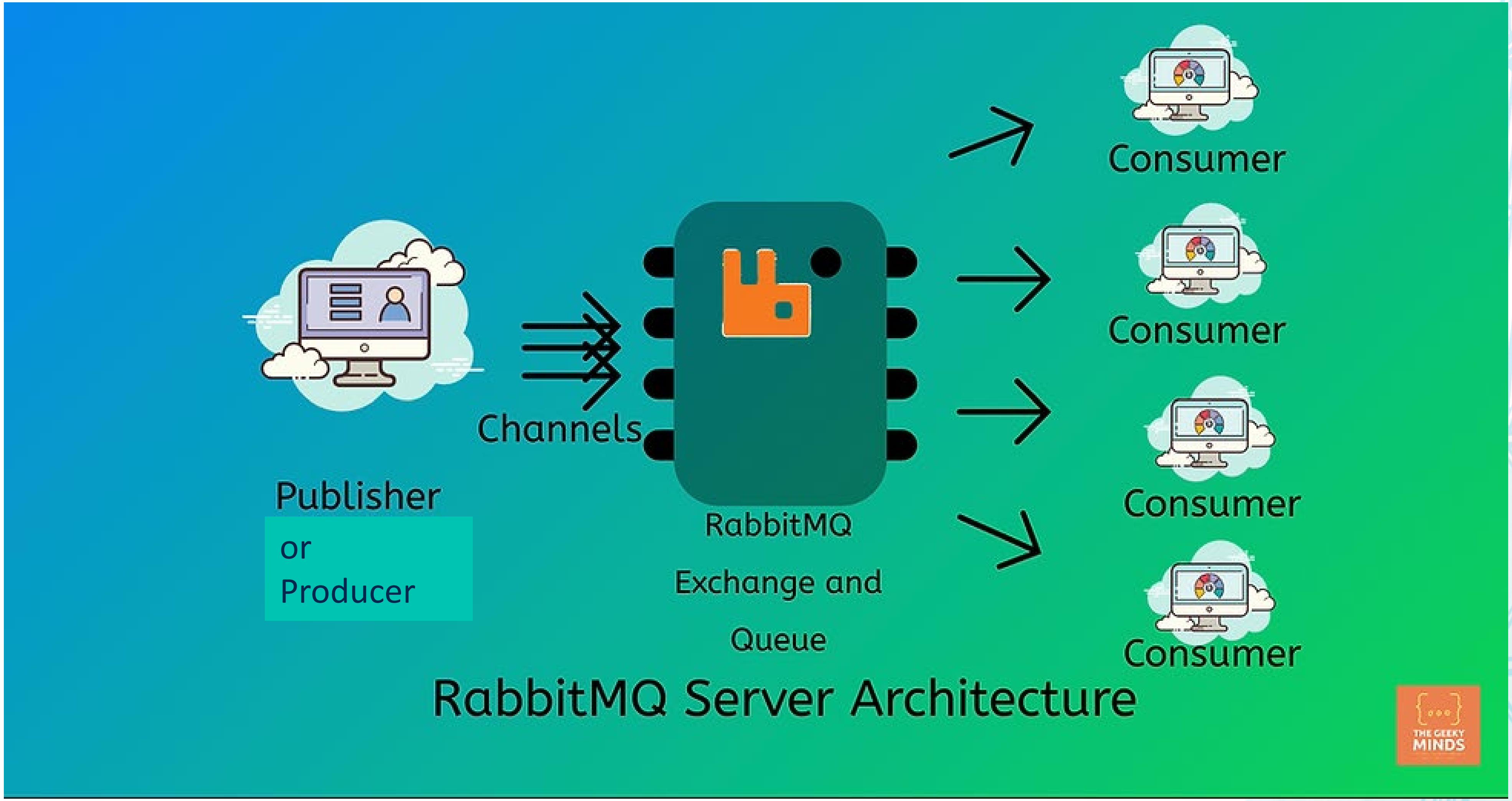
```
// GET Request to Fetch a User denoted by ID

> GET /users/123

// Response

{
    "id": 123,
    "name": "John Doe",
    "email": john.doe@example.com

}
```

Message queuing (MQ)

- Asynchronous communication model
  - Messages sent to a queue and processed by consumers independently of the producer
- Stronger decoupling



RabbitMQ architecture



RabbitMQ demo

- https://www.youtube.com/watch?v=WzO6_4jeliM

RabbitMQ

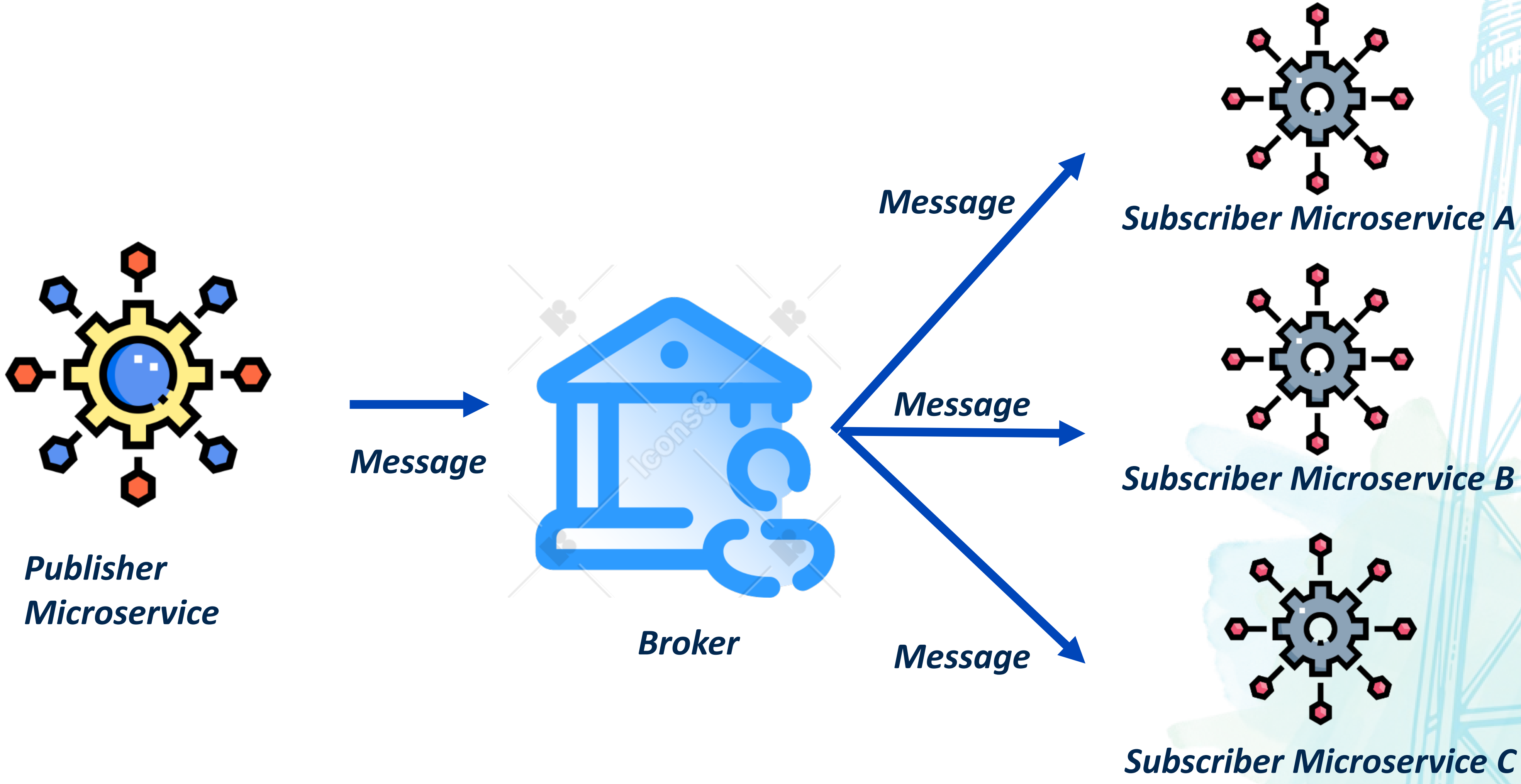
APACHE ACTIVEMQ®

amazon SQS

Pub-sub architecture

- Asynchronous messaging pattern where **publishers** send messages to a central **message broker** or **topic**, and **subscribers** receive messages based on their subscriptions
- Broadcasting: messages can be sent to multiple subscribers
- Typically, messages are persistent at the broker and must be explicitly deleted
- Same frameworks often can act as both MQ or Pub-Sub depending on configuration

Pub-sub architecture



Redis pub-sub demo

- https://www.youtube.com/watch?v=bSe_JBSk5w4

Google Cloud
Pub/Sub

Redis
Pub Sub

Can mix and match!!

- Pipeline architecture using microservices with message queues deployed on a serverless architecture
- Pub-sub architecture with microservices using containers
- Pipeline architecture in one part of the system, with a pub-sub in another
- … and so on
- Pick what is right for your software system!