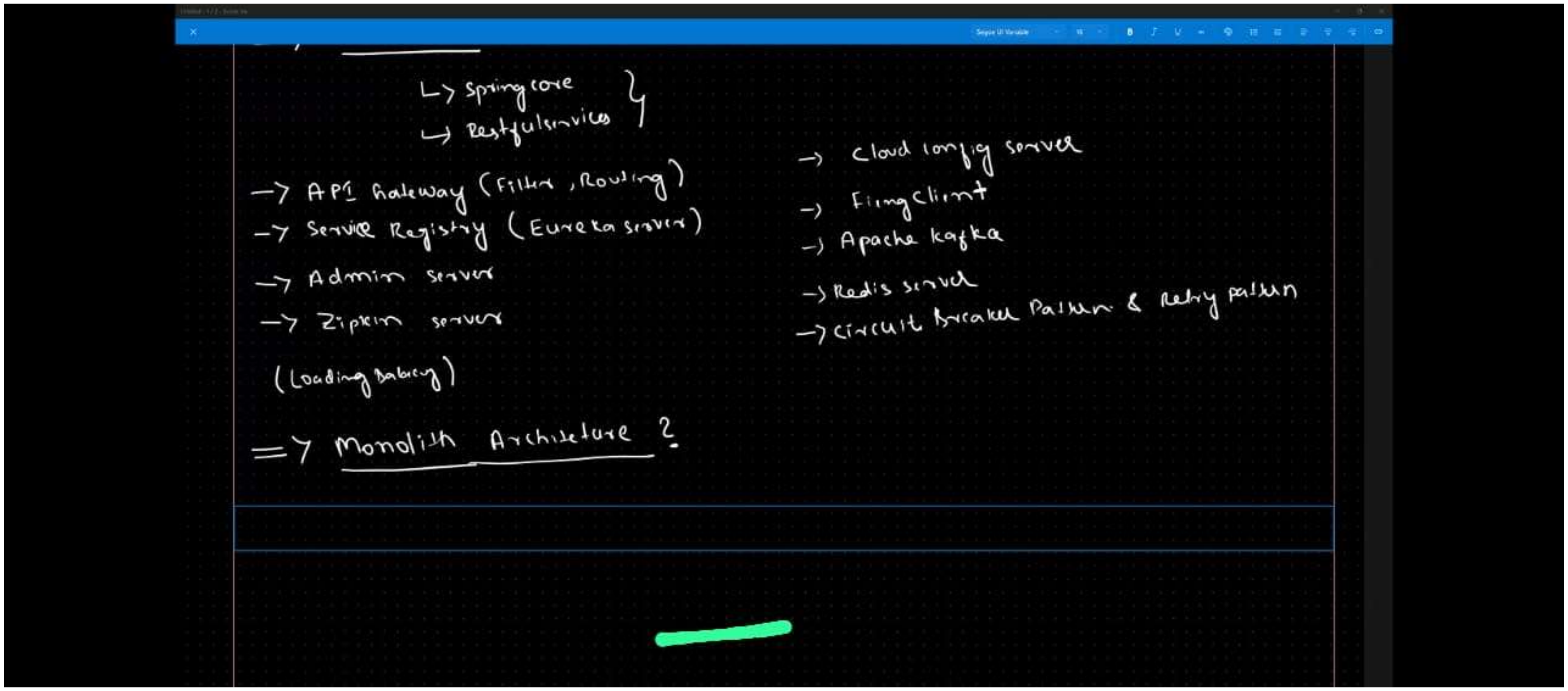
SBMS Introduction



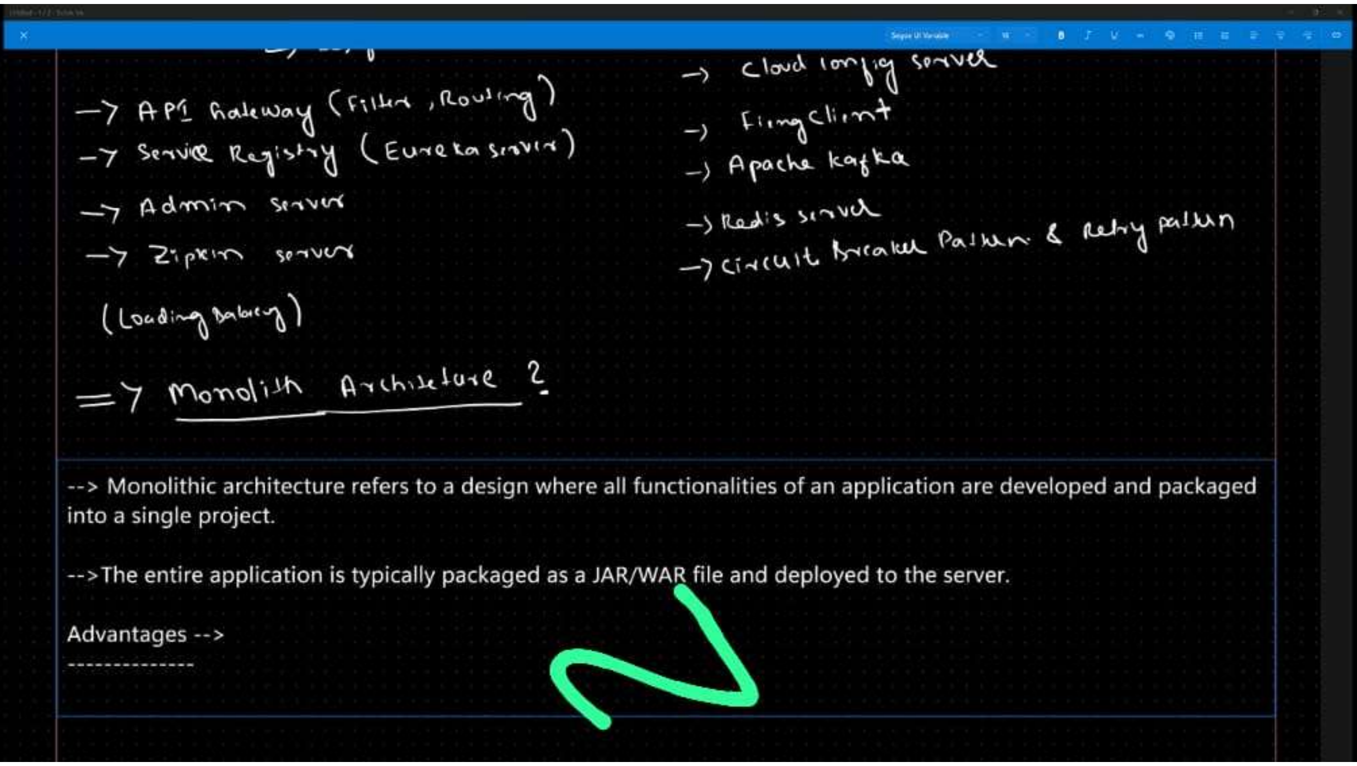
↳ Spring core
↳ Restfulservices }

- → API Gateway (Filter, Routing)
- → Service Registry (Eureka server)
- → Admin server
- → Zipkin server

(Loading Balacing)

⇒ Monolith Architecture ?

- → Cloud config server
- → Fiing Client
- → Apache kafka
- → Redis server
- → Circuit Breaker Pattern & Retry pattern



- → API Gateway (Filter, Routing)
- → Service Registry (Eureka server)
- → Admin server
- → Zipkin server

(Loading balancy)

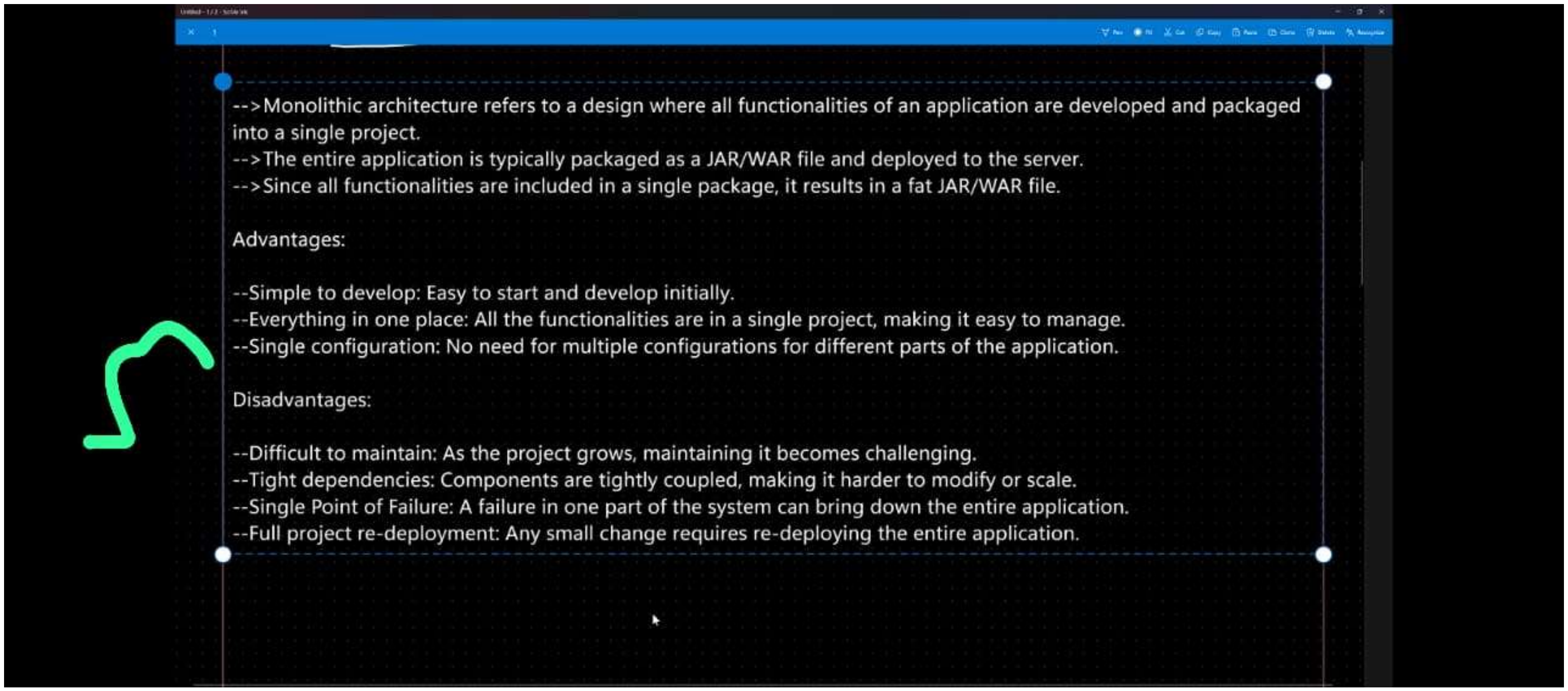
- → Cloud config server
- → FeignClient
- → Apache kafka
- → Redis server
- → Circuit Breaker Pattern & Retry pattern

⇒ Monolith Architecture ?

--> Monolithic architecture refers to a design where all functionalities of an application are developed and packaged into a single project.

-->The entire application is typically packaged as a JAR/WAR file and deployed to the server.

Advantages -->



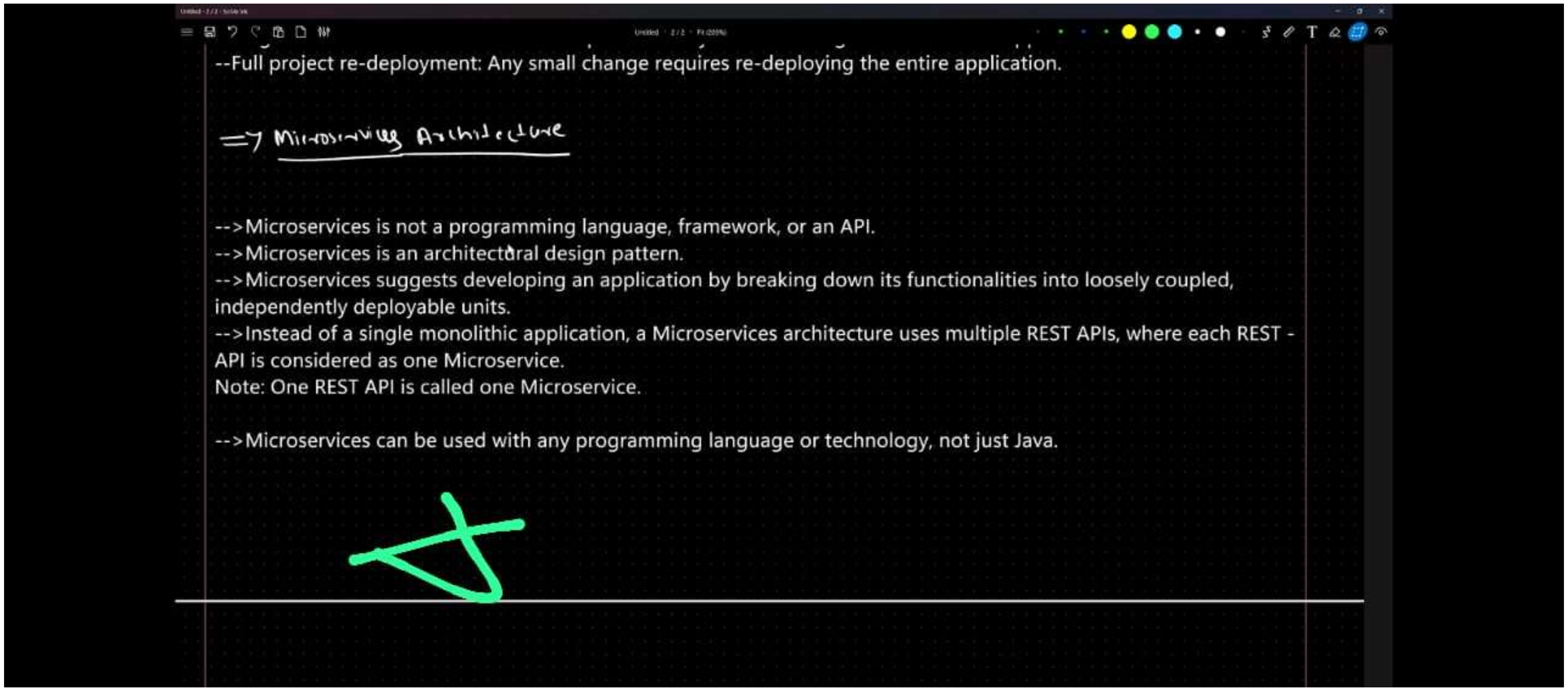
-->Monolithic architecture refers to a design where all functionalities of an application are developed and packaged into a single project.
-->The entire application is typically packaged as a JAR/WAR file and deployed to the server.
-->Since all functionalities are included in a single package, it results in a fat JAR/WAR file.

Advantages:

--Simple to develop: Easy to start and develop initially.
--Everything in one place: All the functionalities are in a single project, making it easy to manage.
--Single configuration: No need for multiple configurations for different parts of the application.

Disadvantages:

--Difficult to maintain: As the project grows, maintaining it becomes challenging.
--Tight dependencies: Components are tightly coupled, making it harder to modify or scale.
--Single Point of Failure: A failure in one part of the system can bring down the entire application.
--Full project re-deployment: Any small change requires re-deploying the entire application.

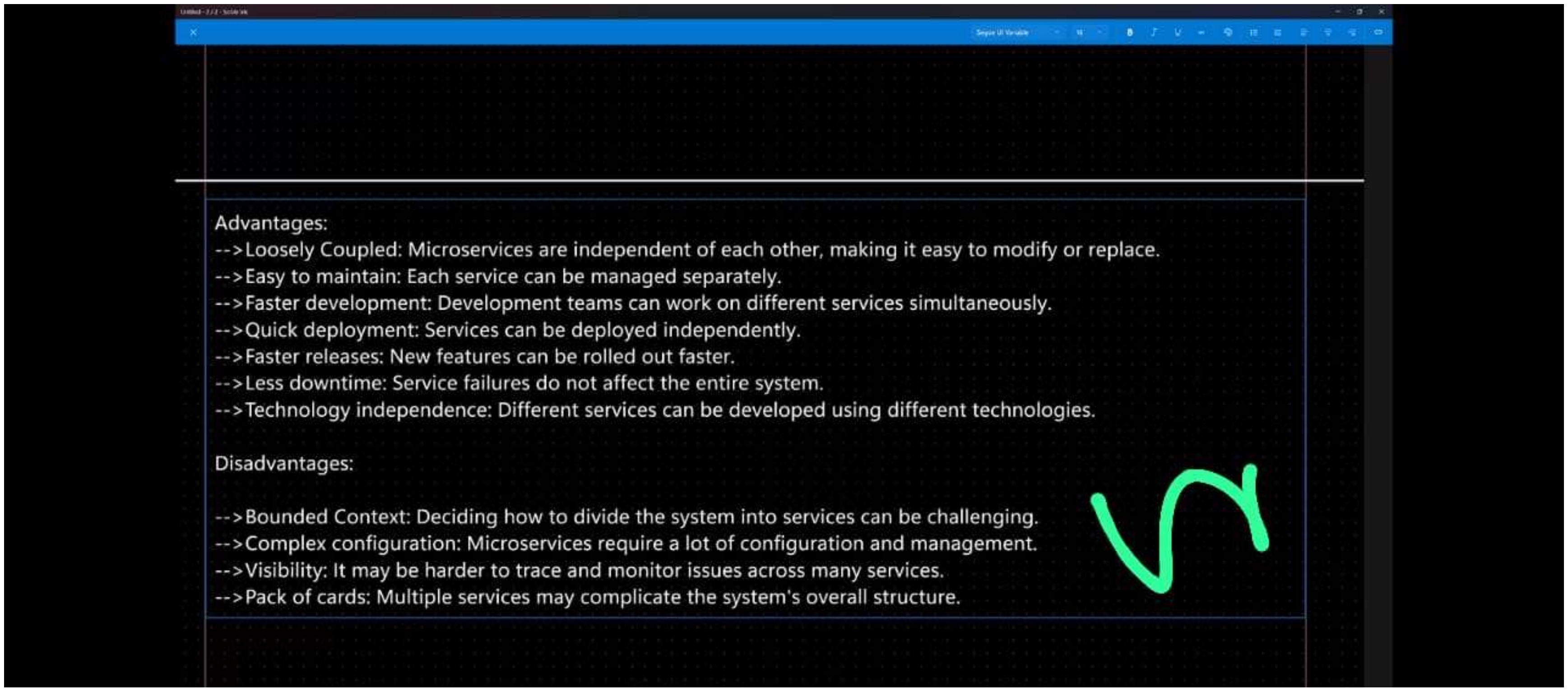


--Full project re-deployment: Any small change requires re-deploying the entire application.

=> Microservices Architecture

-->Microservices is not a programming language, framework, or an API.
-->Microservices is an architectural design pattern.
-->Microservices suggests developing an application by breaking down its functionalities into loosely coupled, independently deployable units.
-->Instead of a single monolithic application, a Microservices architecture uses multiple REST APIs, where each REST - API is considered as one Microservice.
Note: One REST API is called one Microservice.

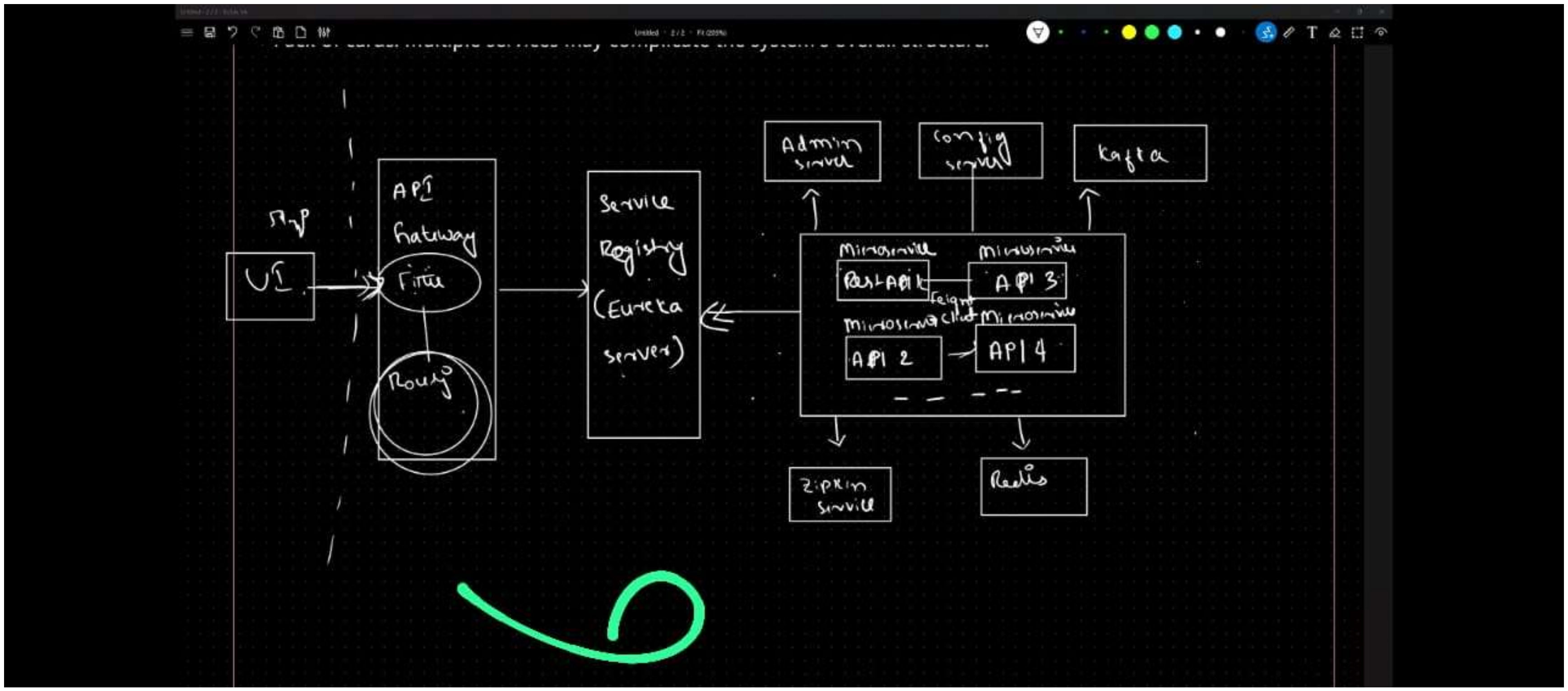
-->Microservices can be used with any programming language or technology, not just Java.



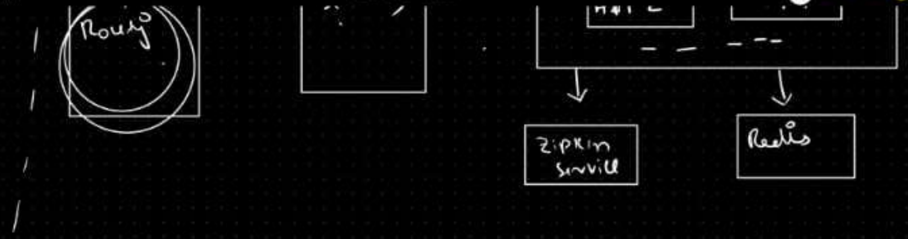
Advantages:
-->Loosely Coupled: Microservices are independent of each other, making it easy to modify or replace.
-->Easy to maintain: Each service can be managed separately.
-->Faster development: Development teams can work on different services simultaneously.
-->Quick deployment: Services can be deployed independently.
-->Faster releases: New features can be rolled out faster.
-->Less downtime: Service failures do not affect the entire system.
-->Technology independence: Different services can be developed using different technologies.

Disadvantages:

-->Bounded Context: Deciding how to divide the system into services can be challenging.
-->Complex configuration: Microservices require a lot of configuration and management.
-->Visibility: It may be harder to trace and monitor issues across many services.
-->Pack of cards: Multiple services may complicate the system's overall structure.



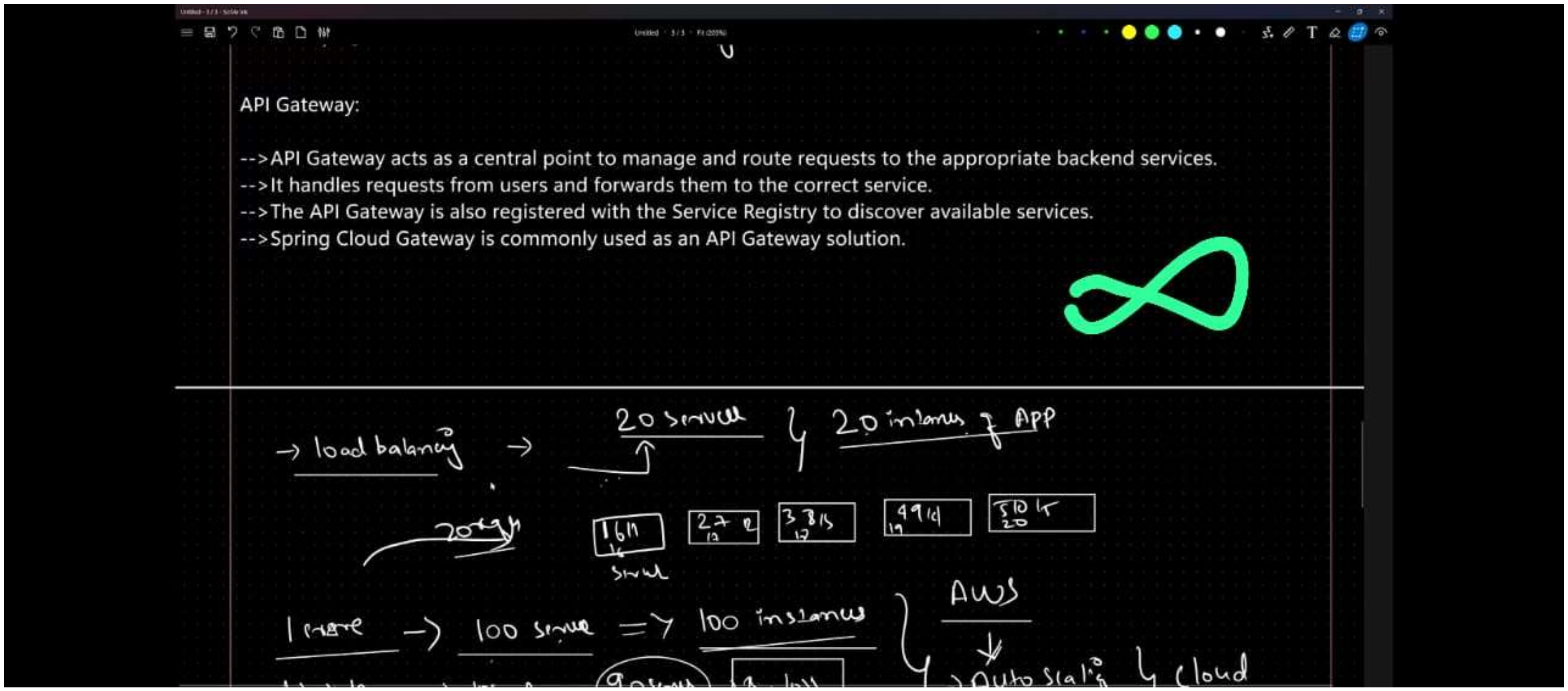
App
UI
API Gateway
Filter
Routing
Service Registry (Eureka server)
Admin server
Config server
Kafka
Microservice
RestAPI 1
Feign client
Microservice
API 3
Microservice
API 2
Microservice
API 4
Zipkin service
Redis



→ <u>Service Registry</u> :- acts as a database that keeps track of All the available services in the system.

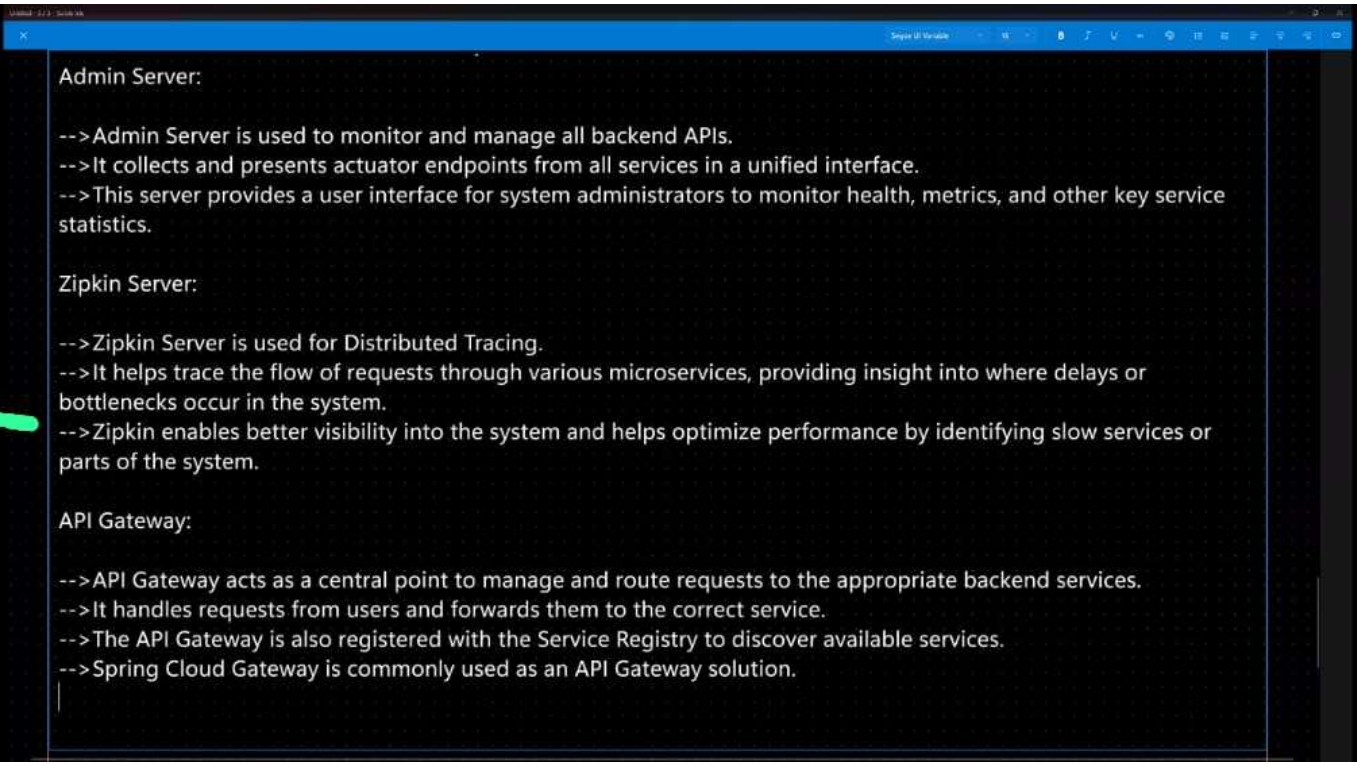
→ It registers each service, allowing the discovery of available services, the number of instances

→ Eureka Server is commonly used as service registry.



API Gateway:

-->API Gateway acts as a central point to manage and route requests to the appropriate backend services.
-->It handles requests from users and forwards them to the correct service.
-->The API Gateway is also registered with the Service Registry to discover available services.
-->Spring Cloud Gateway is commonly used as an API Gateway solution.



Admin Server:

-->Admin Server is used to monitor and manage all backend APIs.
-->It collects and presents actuator endpoints from all services in a unified interface.
-->This server provides a user interface for system administrators to monitor health, metrics, and other key service statistics.

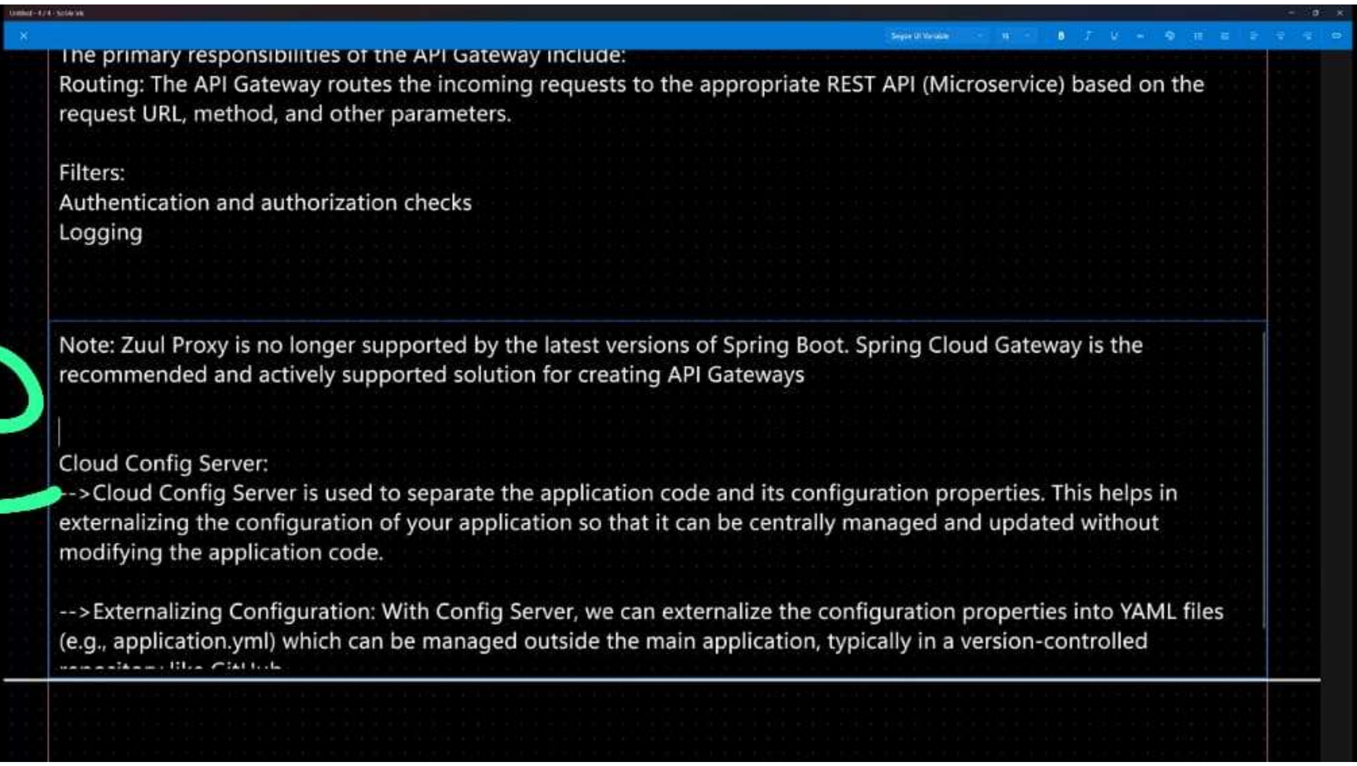
Zipkin Server:

-->Zipkin Server is used for Distributed Tracing.
-->It helps trace the flow of requests through various microservices, providing insight into where delays or bottlenecks occur in the system.
-->Zipkin enables better visibility into the system and helps optimize performance by identifying slow services or parts of the system.

API Gateway:

-->API Gateway acts as a central point to manage and route requests to the appropriate backend services.
-->It handles requests from users and forwards them to the correct service.
-->The API Gateway is also registered with the Service Registry to discover available services.
-->Spring Cloud Gateway is commonly used as an API Gateway solution.

10



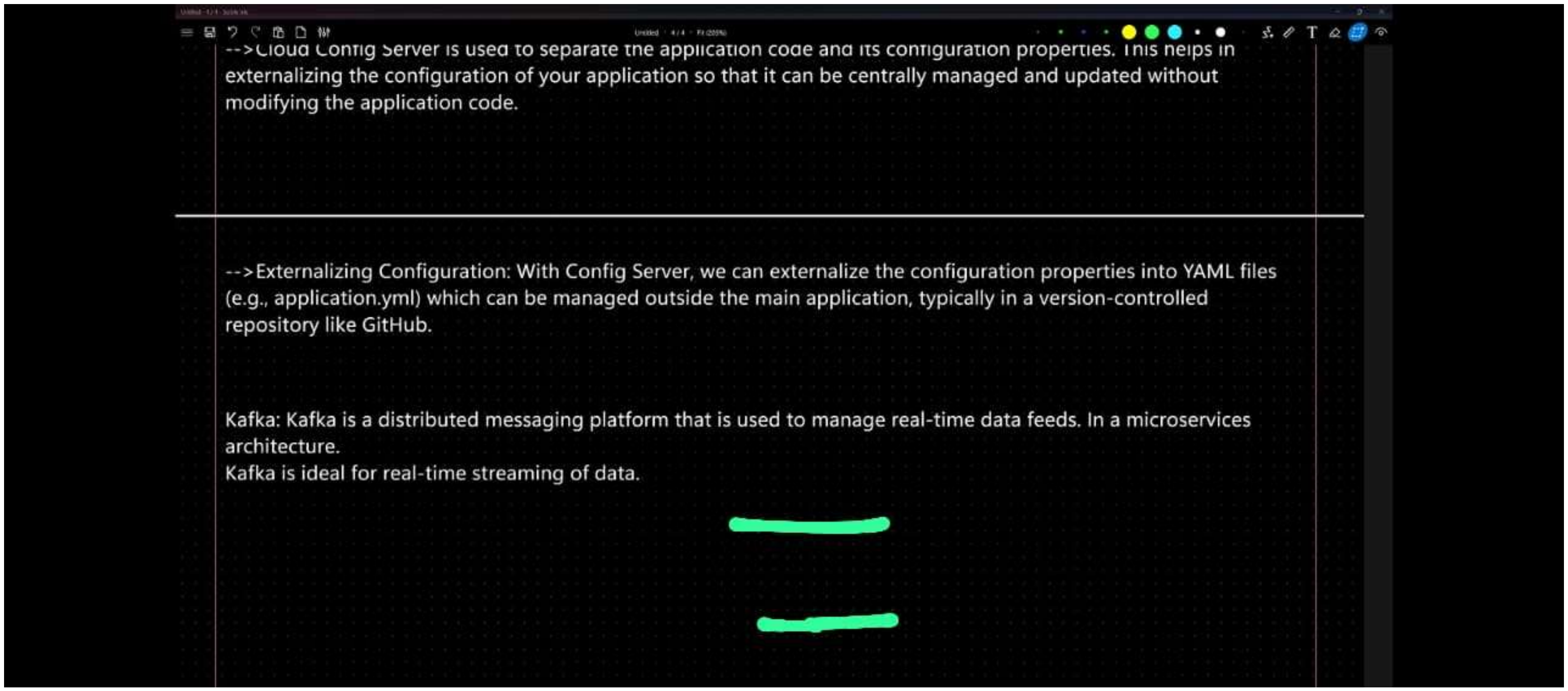
The primary responsibilities of the API Gateway include:
Routing: The API Gateway routes the incoming requests to the appropriate REST API (Microservice) based on the request URL, method, and other parameters.

Filters:
Authentication and authorization checks
Logging

Note: Zuul Proxy is no longer supported by the latest versions of Spring Boot. Spring Cloud Gateway is the recommended and actively supported solution for creating API Gateways

Cloud Config Server:
-->Cloud Config Server is used to separate the application code and its configuration properties. This helps in externalizing the configuration of your application so that it can be centrally managed and updated without modifying the application code.

-->Externalizing Configuration: With Config Server, we can externalize the configuration properties into YAML files (e.g., application.yml) which can be managed outside the main application, typically in a version-controlled



-->Cloud Config Server is used to separate the application code and its configuration properties. This helps in externalizing the configuration of your application so that it can be centrally managed and updated without modifying the application code.

-->Externalizing Configuration: With Config Server, we can externalize the configuration properties into YAML files (e.g., application.yml) which can be managed outside the main application, typically in a version-controlled repository like GitHub.

Kafka: Kafka is a distributed messaging platform that is used to manage real-time data feeds. In a microservices architecture.
Kafka is ideal for real-time streaming of data.