Spring MicroServices

"Microservices are the small services that work together"

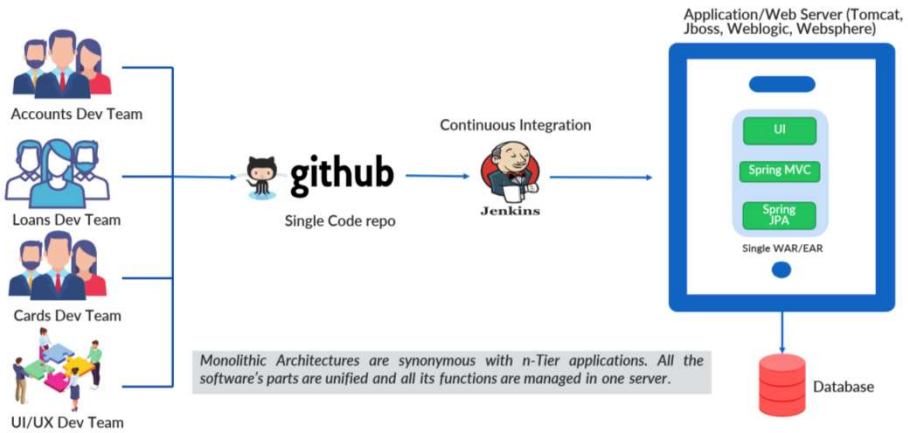
What are Microservices?

- An architectural style that structures an application as a collection of services
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team

Monolith Architecture

- A single Java WAR file.
- A single directory hierarchy of Rails or NodeJS code

Monolithic Architecture



Benefits

- Simple to develop
- Simple to deploy
- Simple to scale

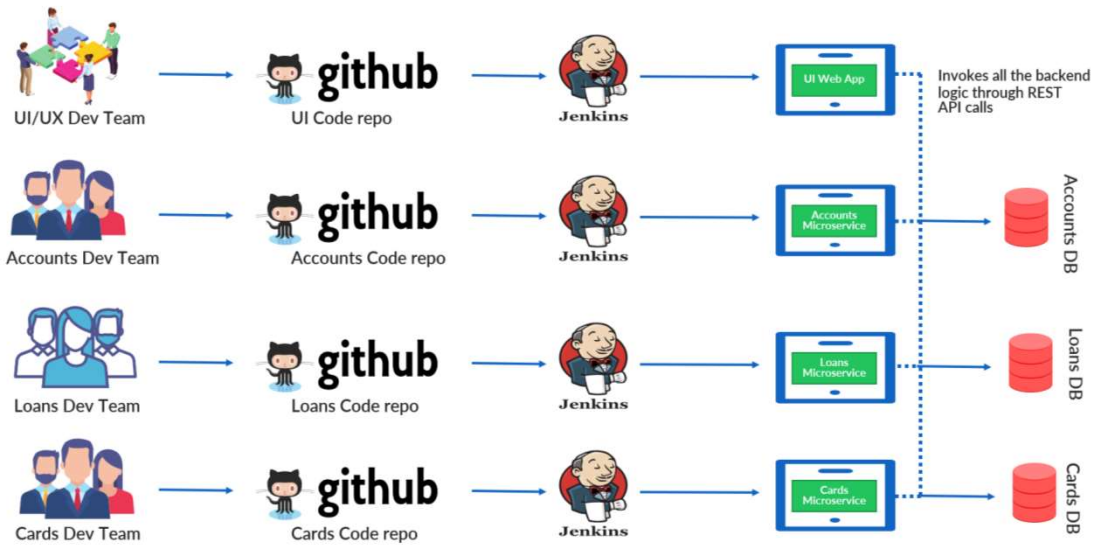
Drawbacks of Monolithic Architecture

- Application is too large and complex
- The size of the application can slow down the start-up time.
- Difficult to scale when different modules have conflicting resource requirements.
- Redeploy the entire application on each update.
- Bug in any module (e.g. memory leak) can potentially bring down the entire process.
- Continuous deployment is difficult.
- Barrier to adopting new technologies.

MicroServices

- Method of developing software applications as a suite of independently deployable, small, modular services.
- Each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.
- Each of these services can be deployed, tweaked, and then redeployed independently.

MicroServices Architecture



Benefits

- Independent development & deployments
- Small, focused teams
- Fault isolation
- Mixed technology stacks

9

Challenges

- Complexity (more moving parts)
- Development and test
- Lack of governance (hard to maintain due to different lang. and frameworks)
- Network congestion and latency (more interservice communication)
- Data integrity (persistence)
- Versioning (updates should not break the service)
- Skillset

10

Best Practices

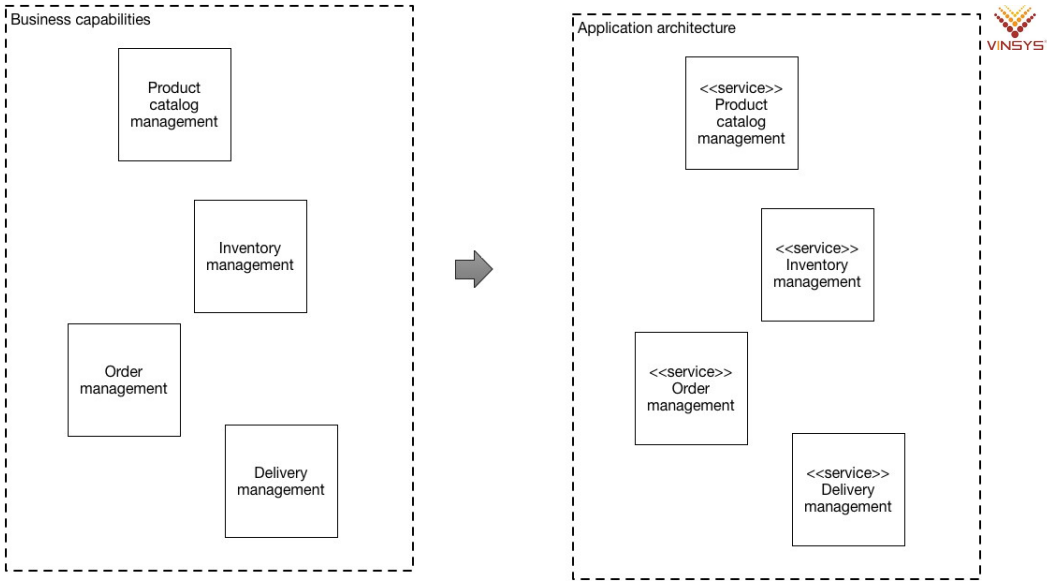
- Model services around the business domain.
- Decentralize everything.
- Individual teams are responsible for designing and building services.
- Avoid sharing code or data schemas.
- Services communicate through well-designed APIs.
- Avoid coupling between services.
  - Causes of coupling include shared database schemas and rigid communication protocols.

11

How to decompose an application into services?

- Business capabilities are often organized into a multi-level hierarchy.
- Define services corresponding to business capabilities.
- A business capability often corresponds to a business
  - Order Management
  - Customer Management
  - Catalog Management
  - Shipping Management

12



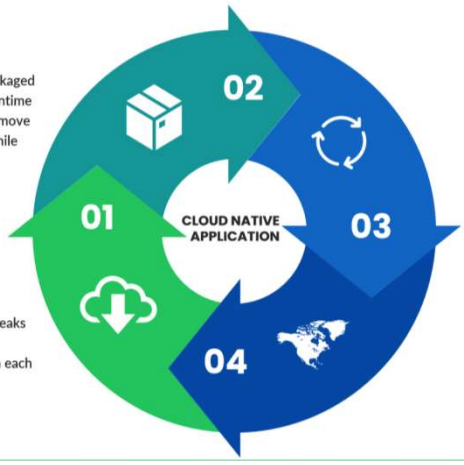
13

Cloud Native Applications

- Cloud-native architecture and technologies are an approach to designing, constructing, and operating workloads that are built in the cloud and take full advantage of the cloud computing model.
- A cloud native application consists of discrete, reusable components known as microservices that are designed to integrate into any cloud environment.
- These microservices act as building blocks and are often packaged in containers.

14

Principles of Cloud Native Apps.



CONTAINERS

Containers allow apps to be packaged and isolated with their entire runtime environment, making it easy to move them between environments while retaining full functionality.

DEVOPS

DevOps is an approach to culture, automation, and platform design intended to deliver increased business value and responsiveness.

MICROSERVICE

A microservices architecture breaks apps down into their smallest components, independent from each other.

CONTINOUS DELIVERY

It's a software development practice in which the process of delivering software is automated to allow short-term deliveries into a production environment.

15

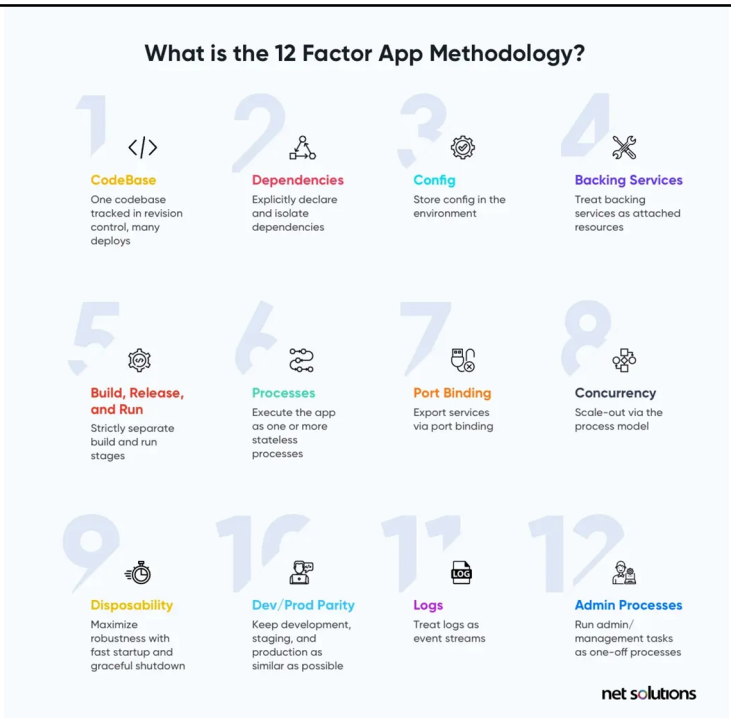
Traditional vs Cloud Native Apps

CLOUD NATIVE APPLICATIONS	TRADITIONAL ENTERPRISE APPLICATIONS
Predictable Behavior	Unpredictable Behavior
OS abstraction	OS dependent
Right-sized capacity & Independent	Oversized capacity & Dependent
Continuous delivery	Waterfall development
Rapid recovery & Automated scalability	Slow recovery

16

12 Factor

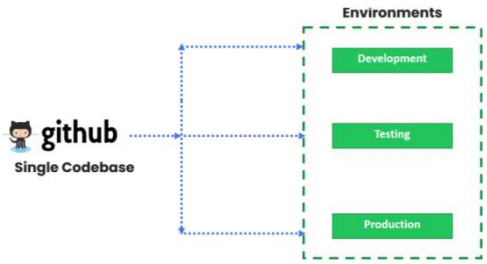
- The twelve-factor app is a methodology for building software-as-a-service apps
- Use declarative formats for setup automation, to minimize time and cost for new developers joining the project
- Suitable for deployment on modern cloud platforms



I. Codebase

- One codebase tracked in revision control

- A twelve-factor app is always tracked in a version control system, such as Git, Mercurial, or Subversion.

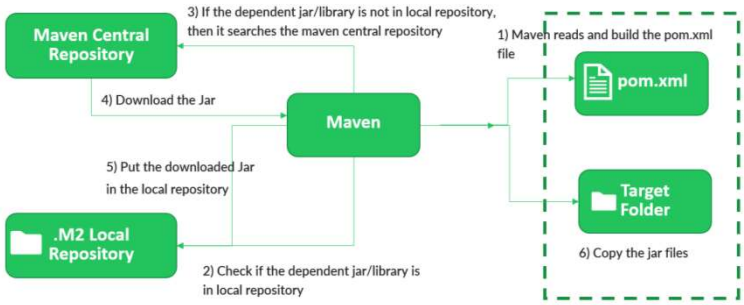


19

II. Dependencies

- Explicitly declare and isolate dependencies

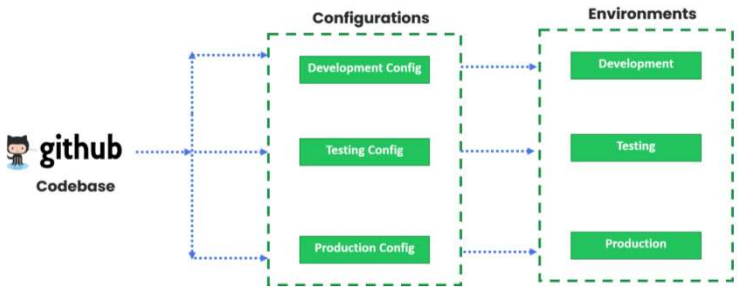
- Declare all dependencies, completely and exactly, via a dependency declaration manifest.



20

III. Config

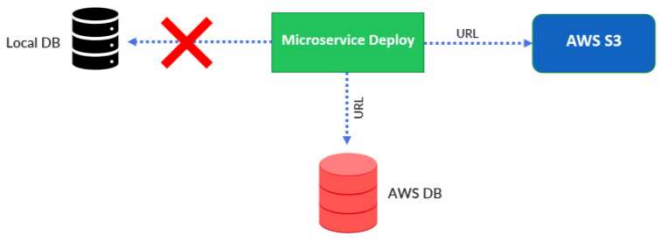
- Store config in the environment



21

IV. Backing services

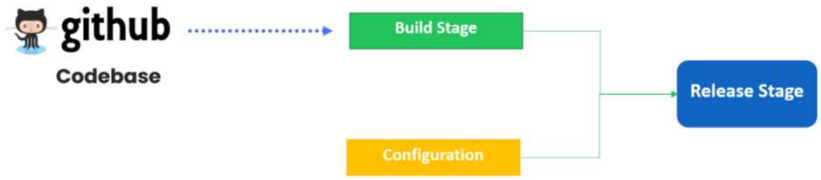
- Treat backing services as attached resources
- A backing service is any service the app consumes over the network as part of its normal operation. Eg. MySQL , RabbitMQ



22

V. Build, release, run

- Strictly separate build and run stages



23

VI. Processes

- Execute the app as one or more stateless processes
- Twelve-factor processes are stateless and share-nothing.
- Any data that needs to persist must be stored in a stateful backing service, typically a database.

24

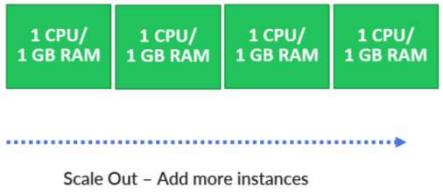
VII. Port binding

- Export services via port binding
- Make sure that your service is visible to others via port binding
  - http://localhost:5000/

25

VIII. Concurrency

- Scale out via the process model
- Small, defined apps allow scaling out as needed to handle the varying loads.
- Each process should be individually scaled



26

IX. Disposability

- Maximize robustness with fast startup and graceful shutdown
- The twelve-factor app's processes are disposable, meaning they can be started or stopped at a moment's notice.
- This facilitates fast elastic scaling, rapid deployment of code or config changes, and robustness of production deploys.

27

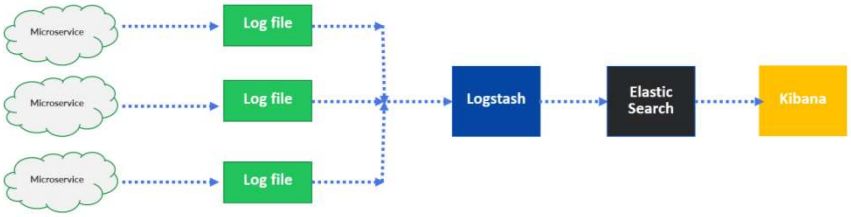
X. Dev/prod parity

- Keep development, staging, and production as similar as possible

28

XI. Logs

- Treat logs as event streams
- Application shouldn't concern itself with the storage of this information.



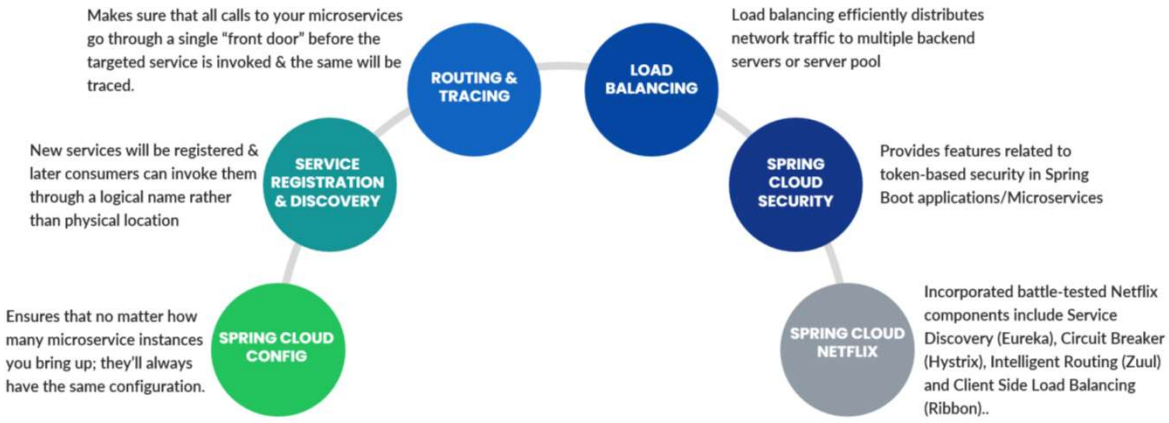
29

XII. Admin processes

- Run admin/management tasks as one-off processes
- Administrative or management tasks should be executed as separate short-lived processes. (scripts)
- The technology ideally supports command execution in a shell that operates on the running environment.

30

Spring Cloud Microservices



RestTemplate

- Synchronous client to perform HTTP requests
- RestTemplate adds the capability of transforming the request and response in JSON or XML to Java objects
- RestTemplate uses the class java.net.HttpURLConnection as the HTTP client
- getForEntity()
- getForObject()
- exchange()

```
Map<String, Integer> params = new HashMap<>();
params.put("id", id);
ResponseEntity<Tax> response =
restTemplate.getForEntity("http://taxservice/taxes/{id}", Tax.class,params);

HttpHeaders headers = new HttpHeaders();
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
HttpEntity <String> entity = new HttpEntity<String>(headers);
restTemplate.exchange("http://localhost:8080/products", HttpMethod.GET, entity,
String.class).getBody();
```

33

```
HttpHeaders headers = new HttpHeaders();
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);
restTemplate.exchange("http://localhost:8080/products",
HttpMethod.POST, entity, String.class).getBody();

HttpHeaders headers = new HttpHeaders();
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);
restTemplate.exchange("http://localhost:8080/products/"+id, HttpMethod.PUT,
entity, String.class).getBody();
```

34

```
HttpHeaders headers = new HttpHeaders();
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
HttpEntity<Product> entity = new HttpEntity<Product>(headers);
restTemplate.exchange("http://localhost:8080/products/"+id, HttpMethod.DELETE,
entity, String.class).getBody();
```


Spring Cloud OpenFeign

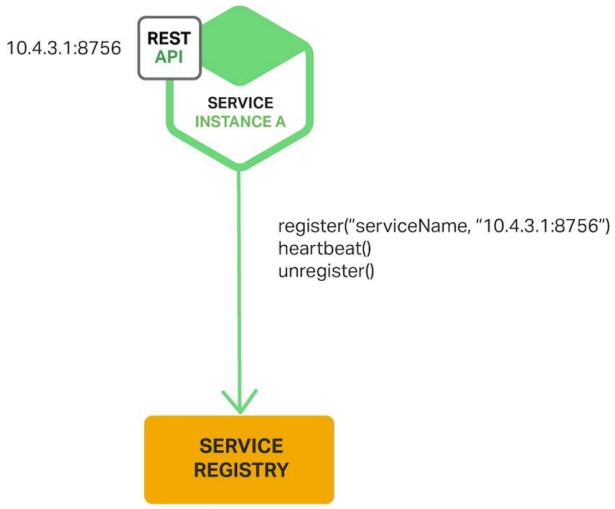
- Feign is a declarative web service client.
- Makes writing web service clients easier.

```
@SpringBootApplication
@EnableFeignClients public class Application {
public static void main(String[] args)
{
SpringApplication.run(Application.class, args);
}
}
```

```
@FeignClient("stores")
public interface StoreClient {
    @RequestMapping(method = RequestMethod.GET, value = "/stores")
    List<Store> getStores();

    @RequestMapping(method = RequestMethod.GET, value = "/stores")
    Page<Store> getStores(Pageable pageable);

    @RequestMapping(method = RequestMethod.POST, value = "/stores/{storeId}", consumes = "application/json")
    Store update(@PathVariable("storeId") Long storeId, Store store);

    @RequestMapping(method = RequestMethod.DELETE, value = "/stores/{storeId:\\d+}")
    void delete(@PathVariable Long storeId);
}
```

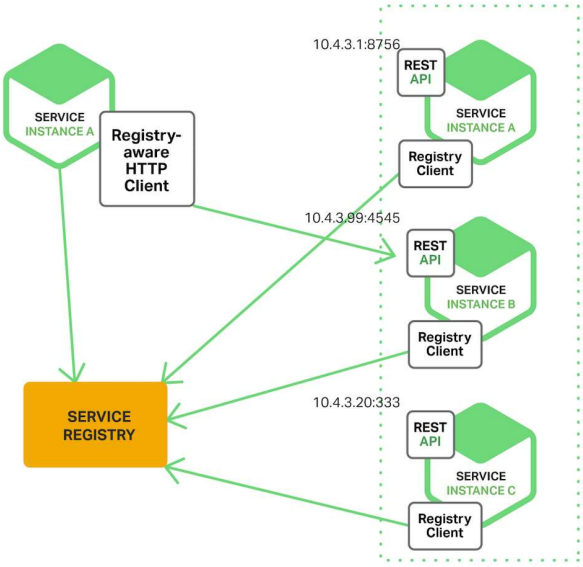
Service Registry : Eureka Server

- The service registry is a key part of service discovery.
- Database containing the network locations of service instances.
- A service registry needs to be highly available and up to date.
- Netflix Eureka is a service registry.



39

Service Discovery : Eureka Client

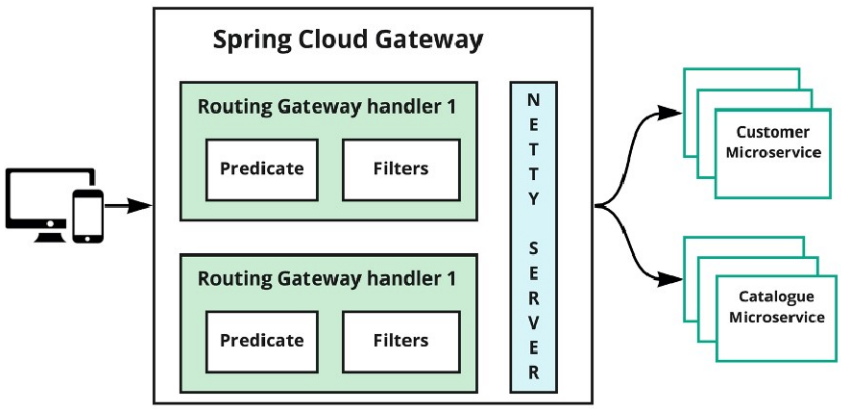


40

Spring Cloud Gateway

- Spring Cloud Gateway provides a library for building API gateways on top of Spring and Java.
- Spring Cloud Gateway is intended to sit between a requester and a resource that's being requested, where it intercepts, analyzes, and modifies every request.
- Provides a flexible way of routing requests based on a number of criteria, as well as focuses on cross-cutting concerns such as security, resiliency, and monitoring.

41



42

Spring Cloud Config

- Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system.
- A central place to manage external properties for applications across all environments.

43

Spring Cloud Circuit Breaker

- Spring Cloud Circuit breaker provides an abstraction across different circuit breaker implementations.
- **Circuit Breaker -** temporary blocks possible failures
- **Retry -** repeats failed executions
- **Rate Limiter -** limits executions/period

44