Resilient Strategies for fintech

Dmitrii Pakhomov

About me

Dmitrii Pakhomov

- Software architect with 10 years of experience.
- I build mission critical Fintech system handling extremely high load.
- Author of many scala libraries for resilience and observability

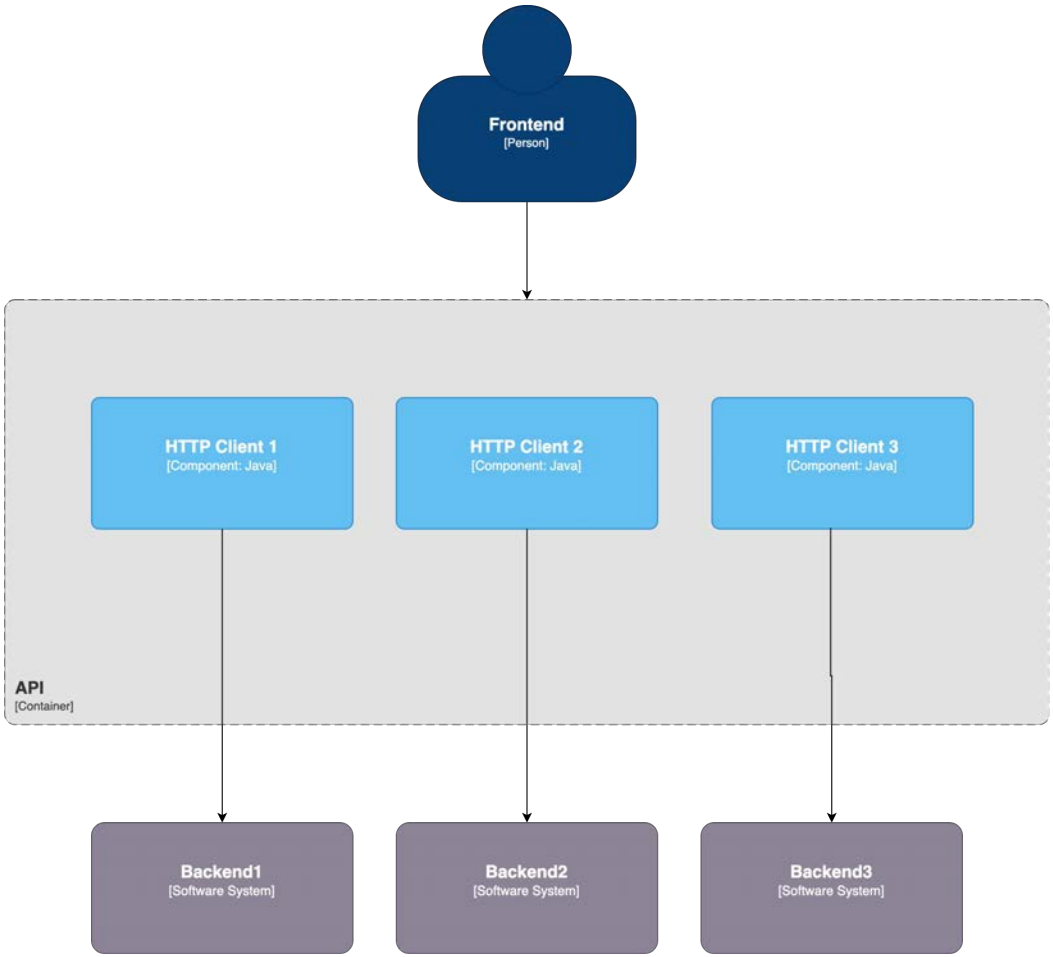
What is Resilience?

- Retry
- Circuit Breaker
- Bulkhead
- Cache
- Fallback
- Reloadable config

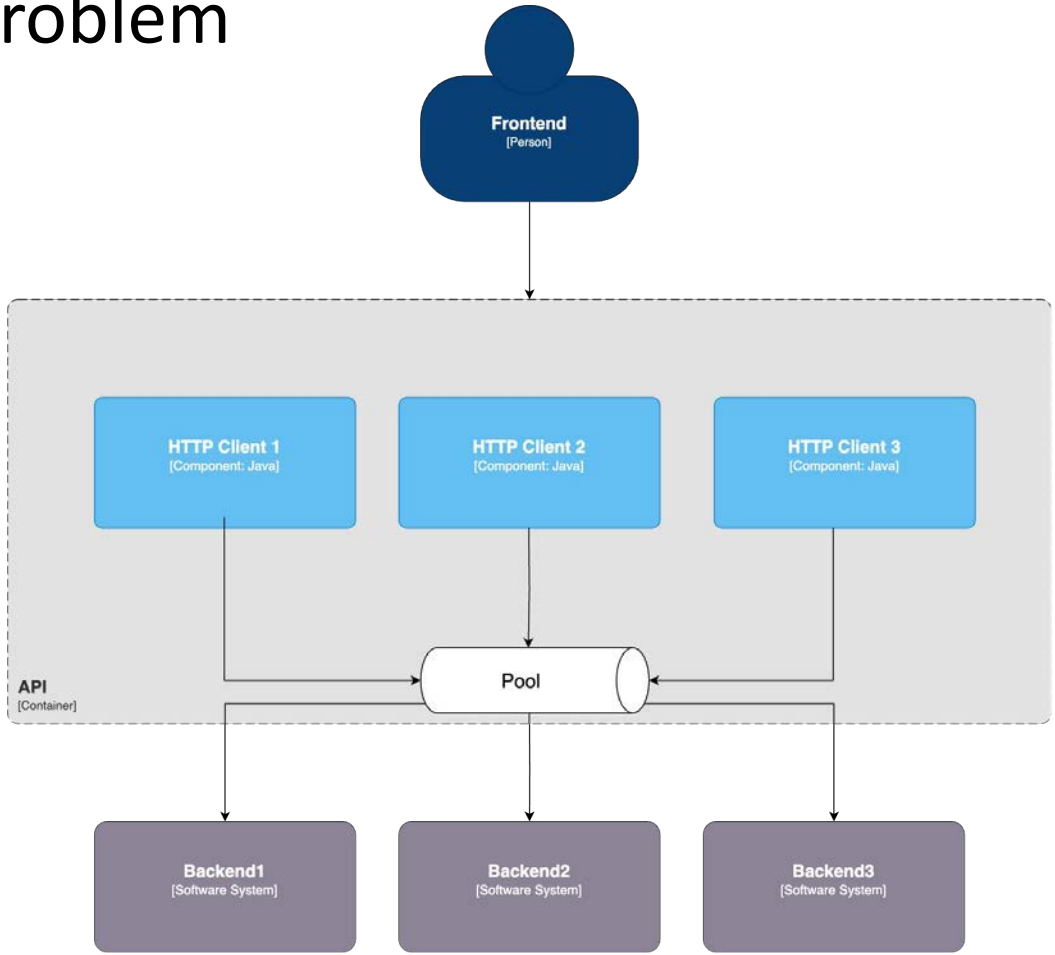
- Fault Tolerance
- Graceful Degradation
- Failure Detection and Recovery
- Isolation and Containment

Bulkhead

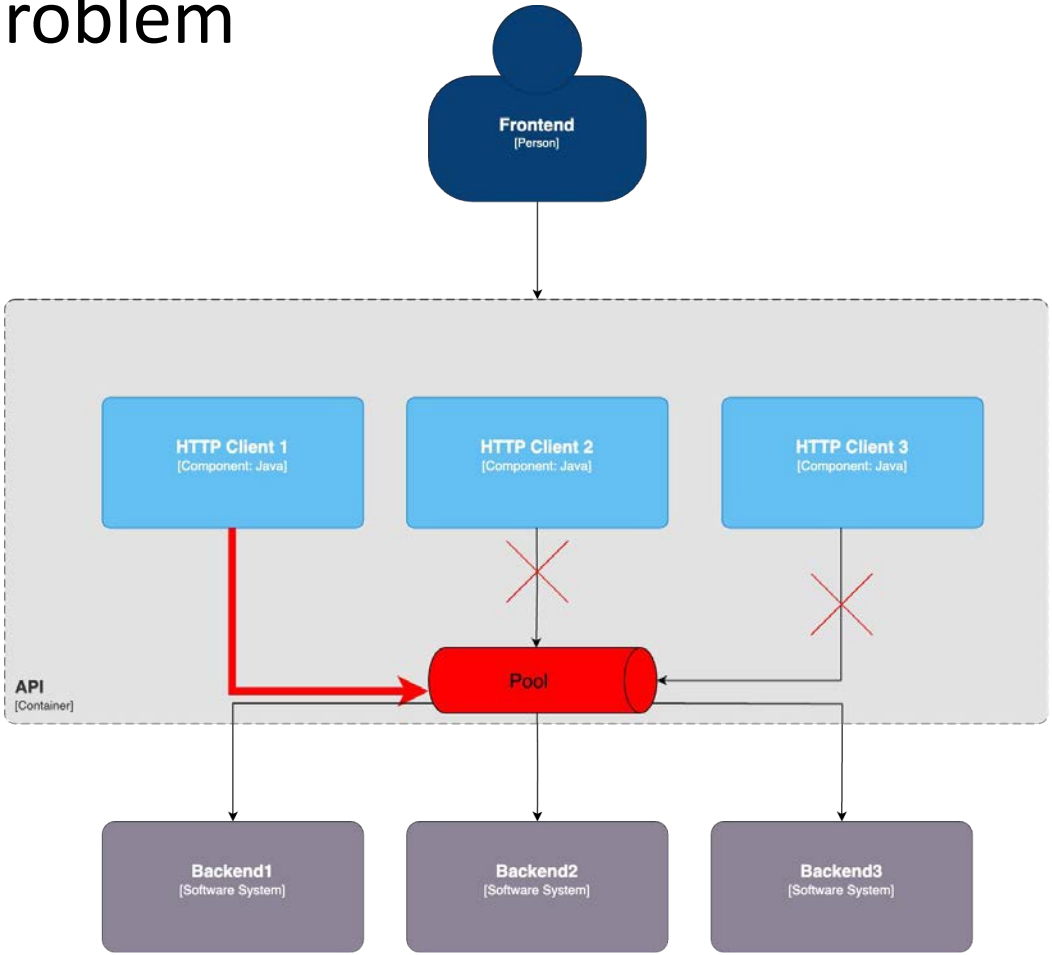
Bulkhead


Frontend
[Person]
HTTP Client 1
[Component: Java]
HTTP Client 2
[Component: Java]
HTTP Client 3
[Component: Java]
API
[Container]
Backend1
[Software System]
Backend2
[Software System]
Backend3
[Software System]

Bulkhead: Problem

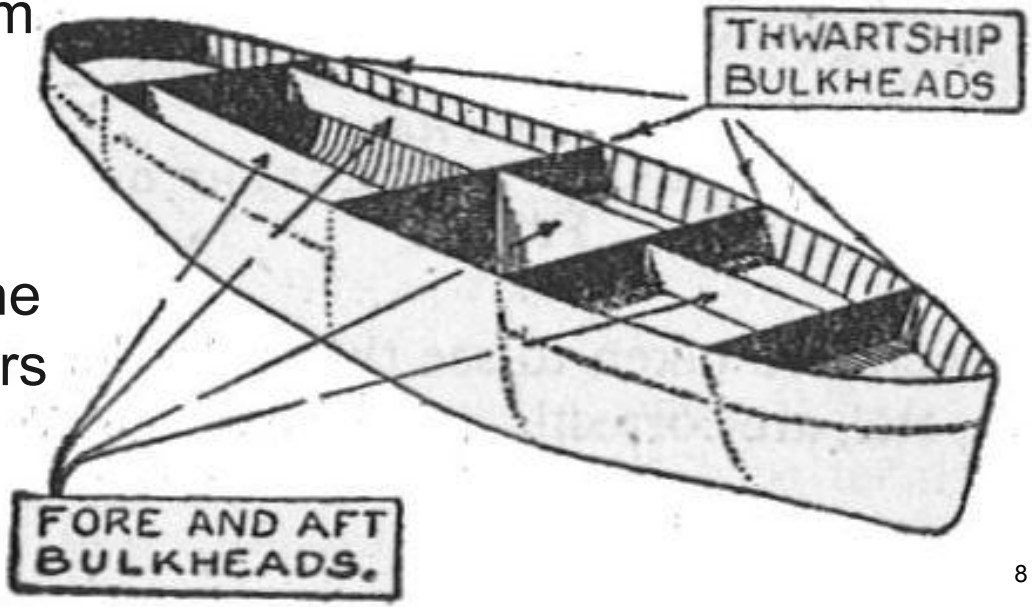


Bulkhead: Problem

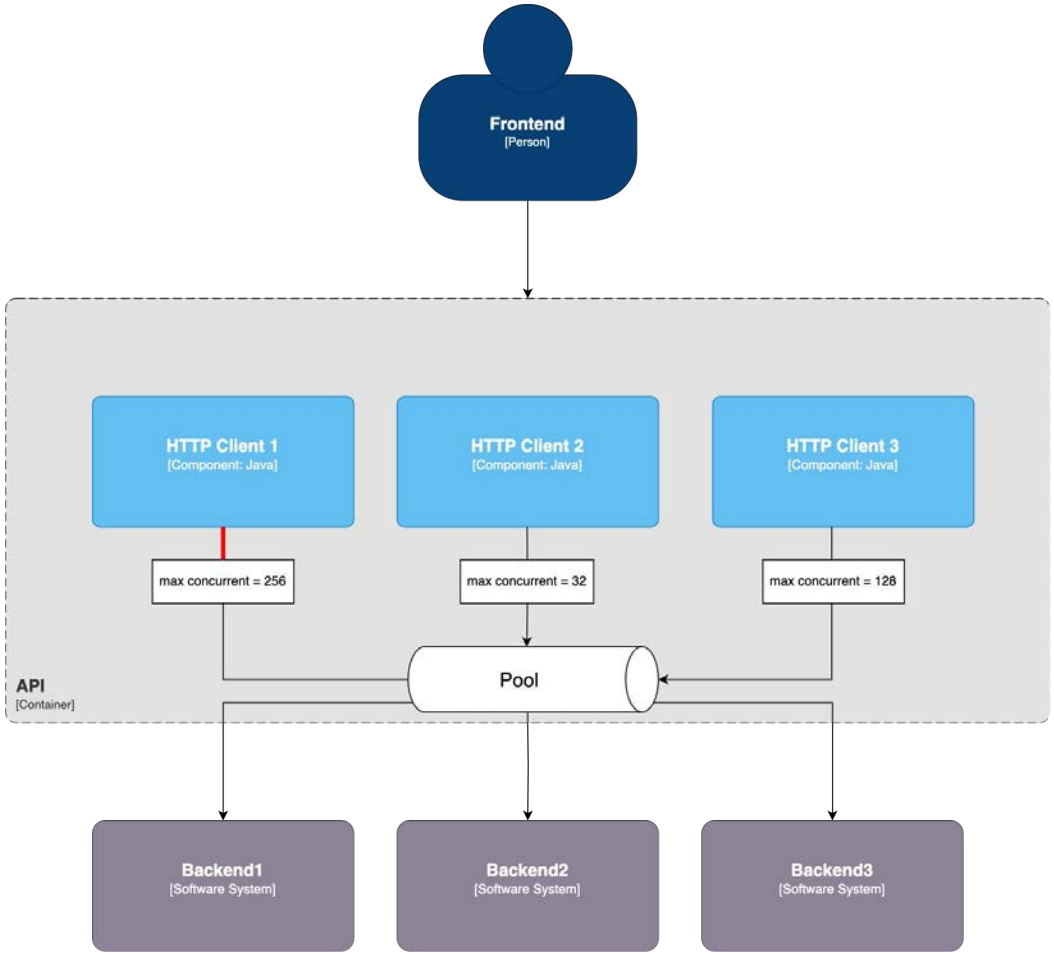


Bulkhead

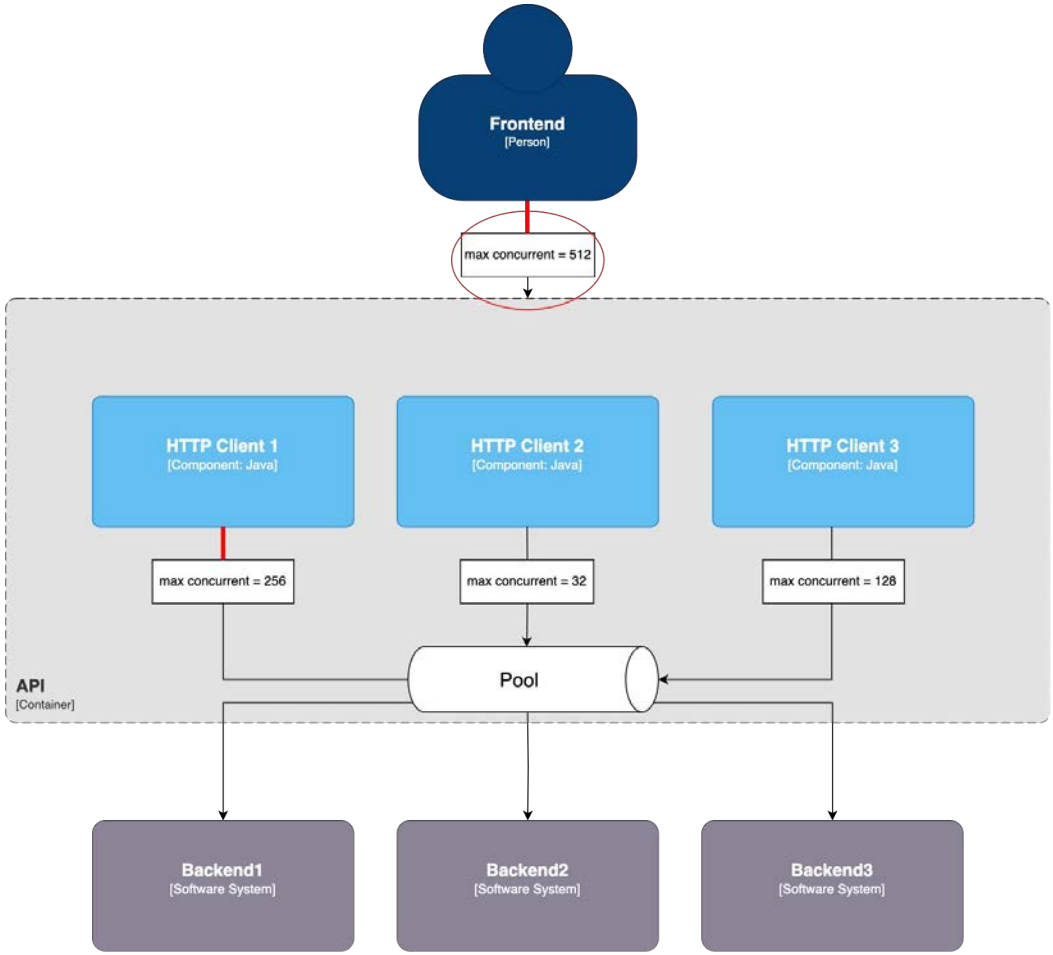
- Isolating resource pools
- This isolation prevents failures in one part of the system from cascading to other parts
- It's like having watertight compartments in a ship: if one compartment leaks, the others stay dry.



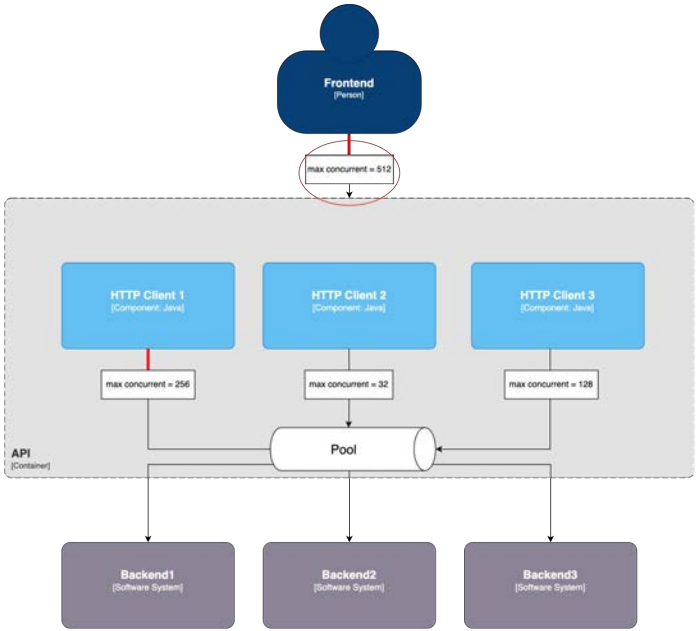
Bulkhead


Frontend
[Person]
HTTP Client 1
[Component: Java]
HTTP Client 2
[Component: Java]
HTTP Client 3
[Component: Java]
max concurrent = 256
max concurrent = 32
max concurrent = 128
Pool
API
[Container]
Backend1
[Software System]
Backend2
[Software System]
Backend3
[Software System]

Bulkhead


Frontend
[Person]
max concurrent = 512
HTTP Client 1
[Component: Java]
HTTP Client 2
[Component: Java]
HTTP Client 3
[Component: Java]
max concurrent = 256
max concurrent = 32
max concurrent = 128
Pool
API
[Container]
Backend1
[Software System]
Backend2
[Software System]
Backend3
[Software System]

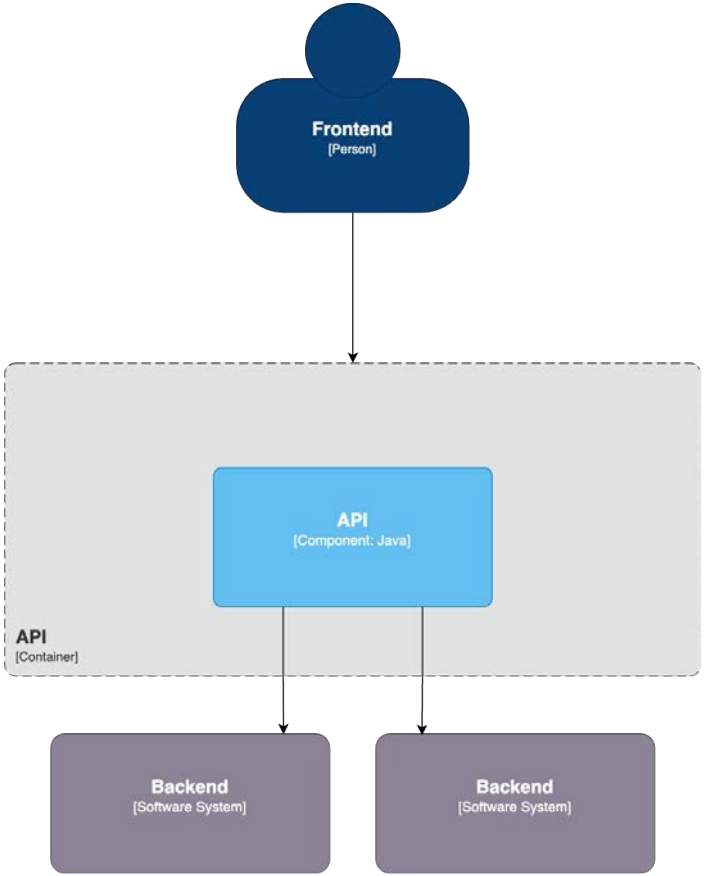
Bulkhead



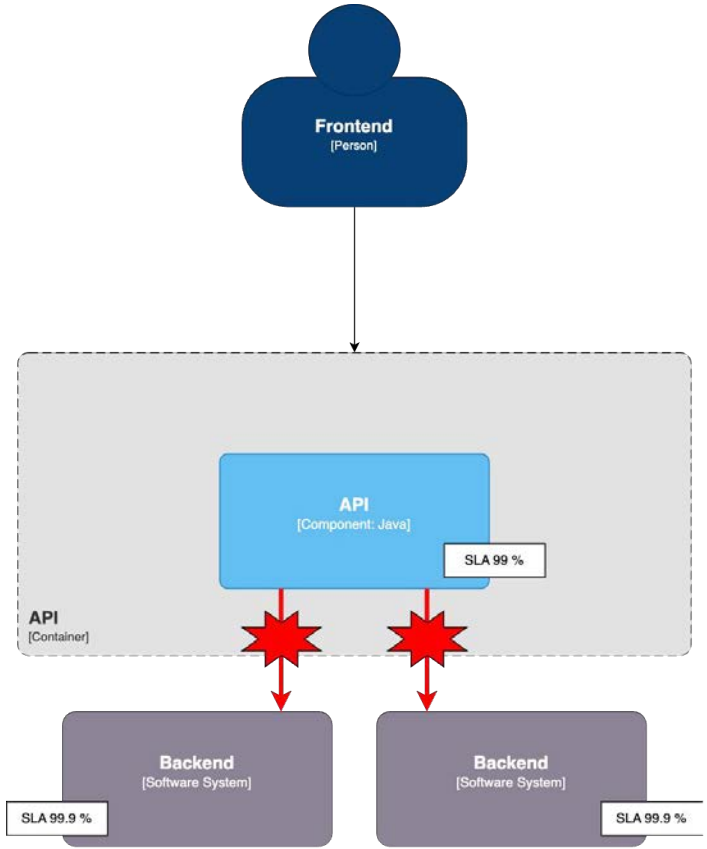

```scala
class Bulkhead(maxConcurrent: Int, maxQueueSize: Int) {
  private val concurrentSemaphore = Semaphore.make(maxConcurrent)
  private val queueSemaphore = Semaphore.make(maxQueueSize)

  def execute[A](task: Task[A]): Task[A] =
    for {
      _ <- queueSemaphore.tryAcquire
      _ <- concurrentSemaphore.acquire
      result <- task
        .ensuring(concurrentSemaphore.release)
        .ensuring(queueSemaphore.release)
    } yield result
}

// Example usage
val bulkhead = new Bulkhead(maxConcurrent = 32, maxQueueSize = 16)
val myTask: Task[String] = ???
val result = bulkhead.execute(myTask)
```

Cache

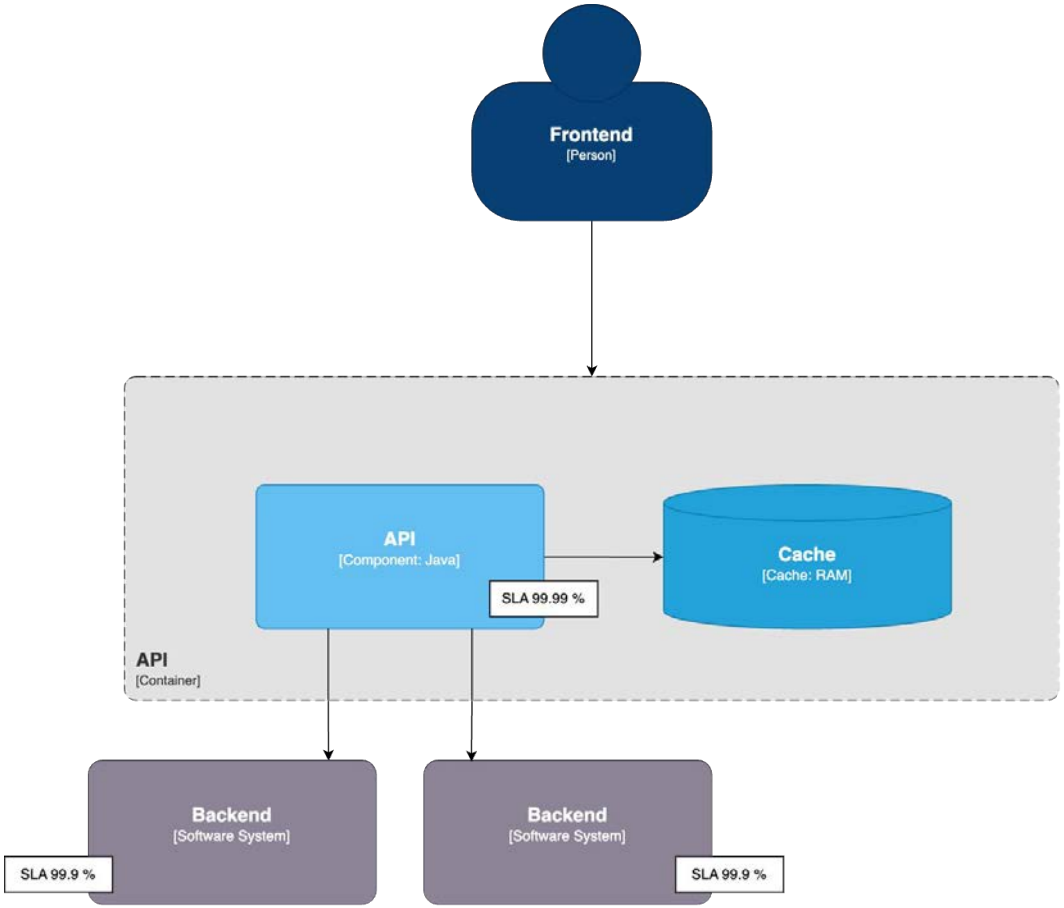
Cache: Problem



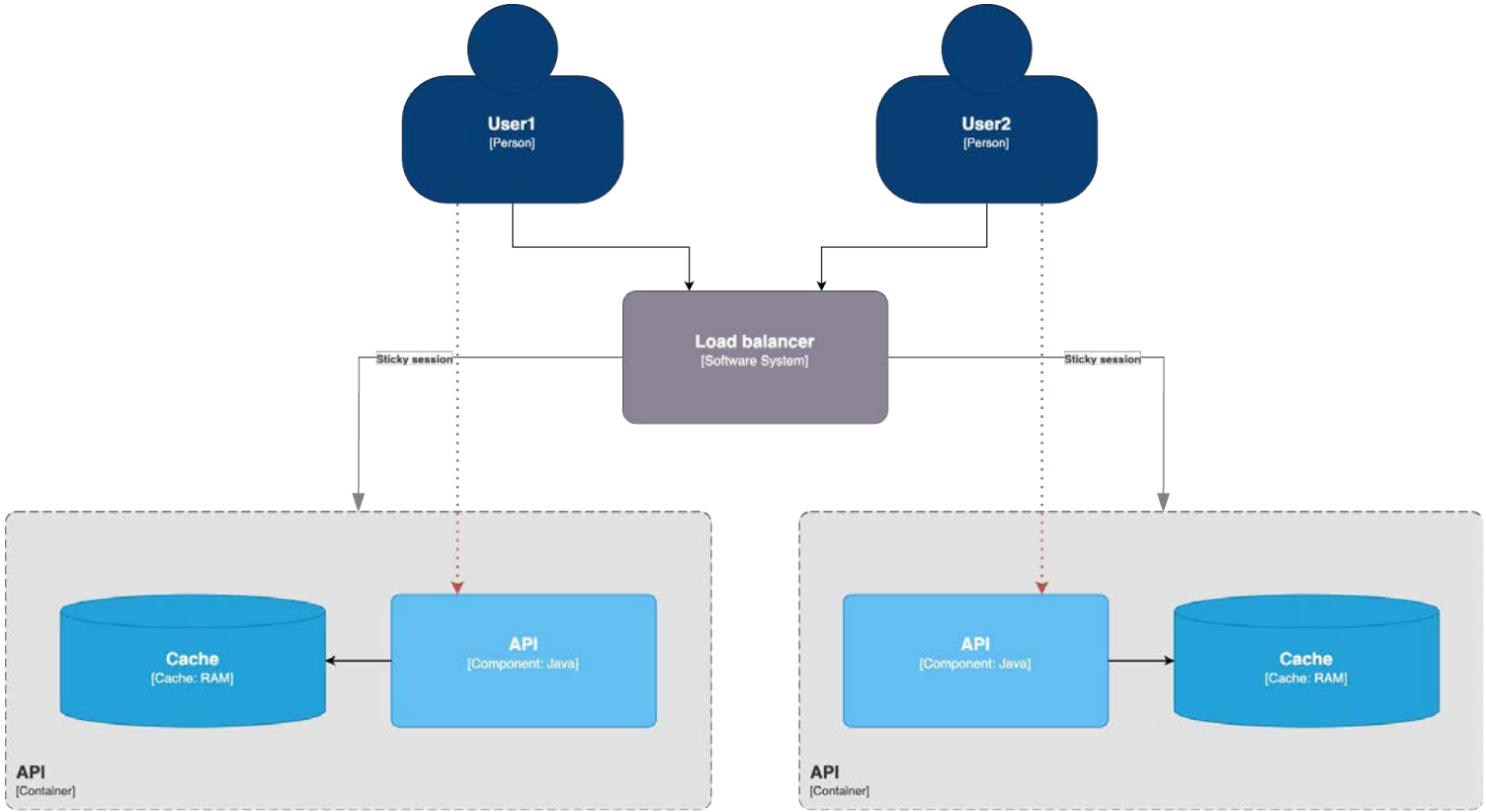
Cache: Problem



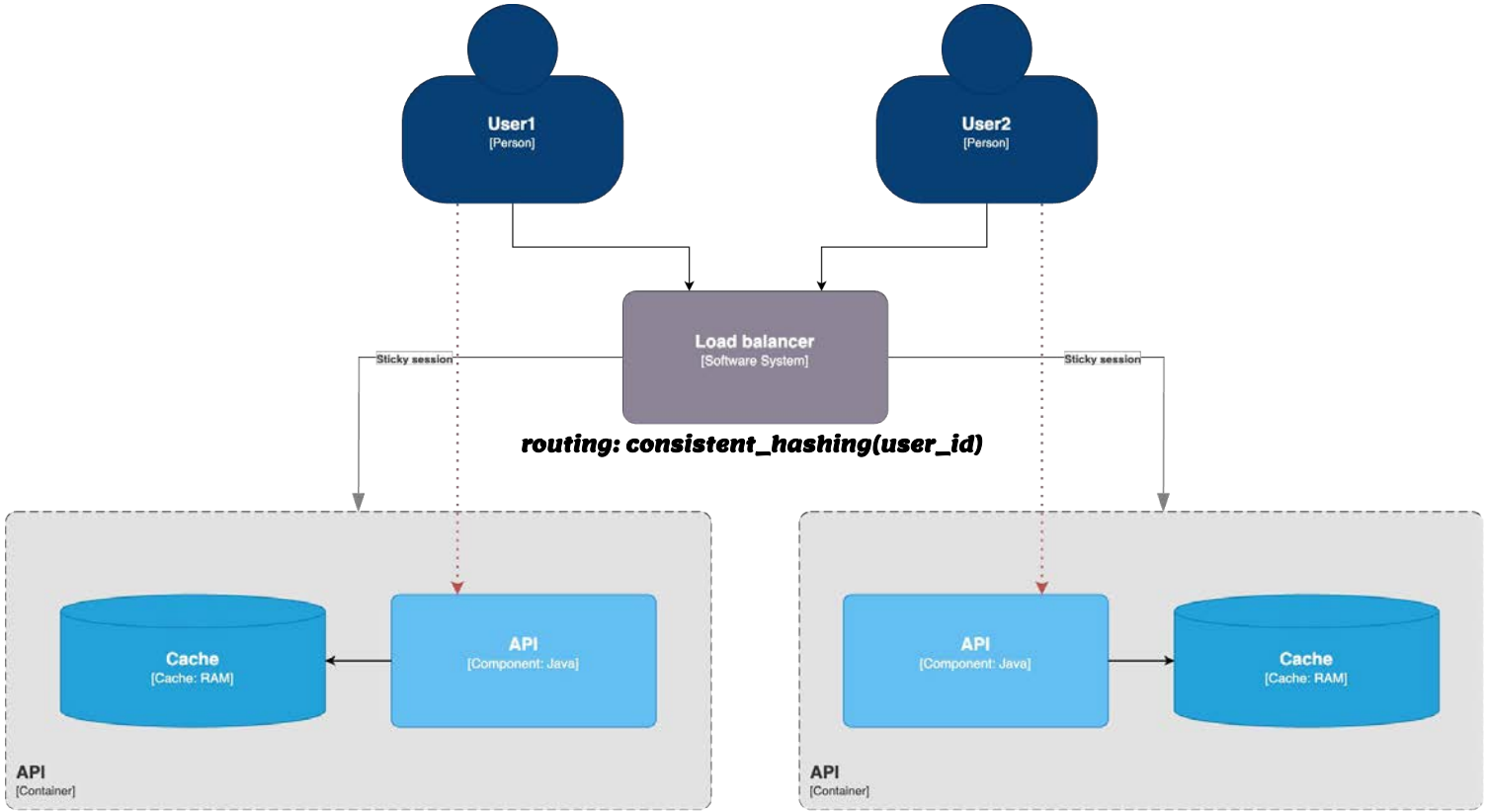
Inmemory cache



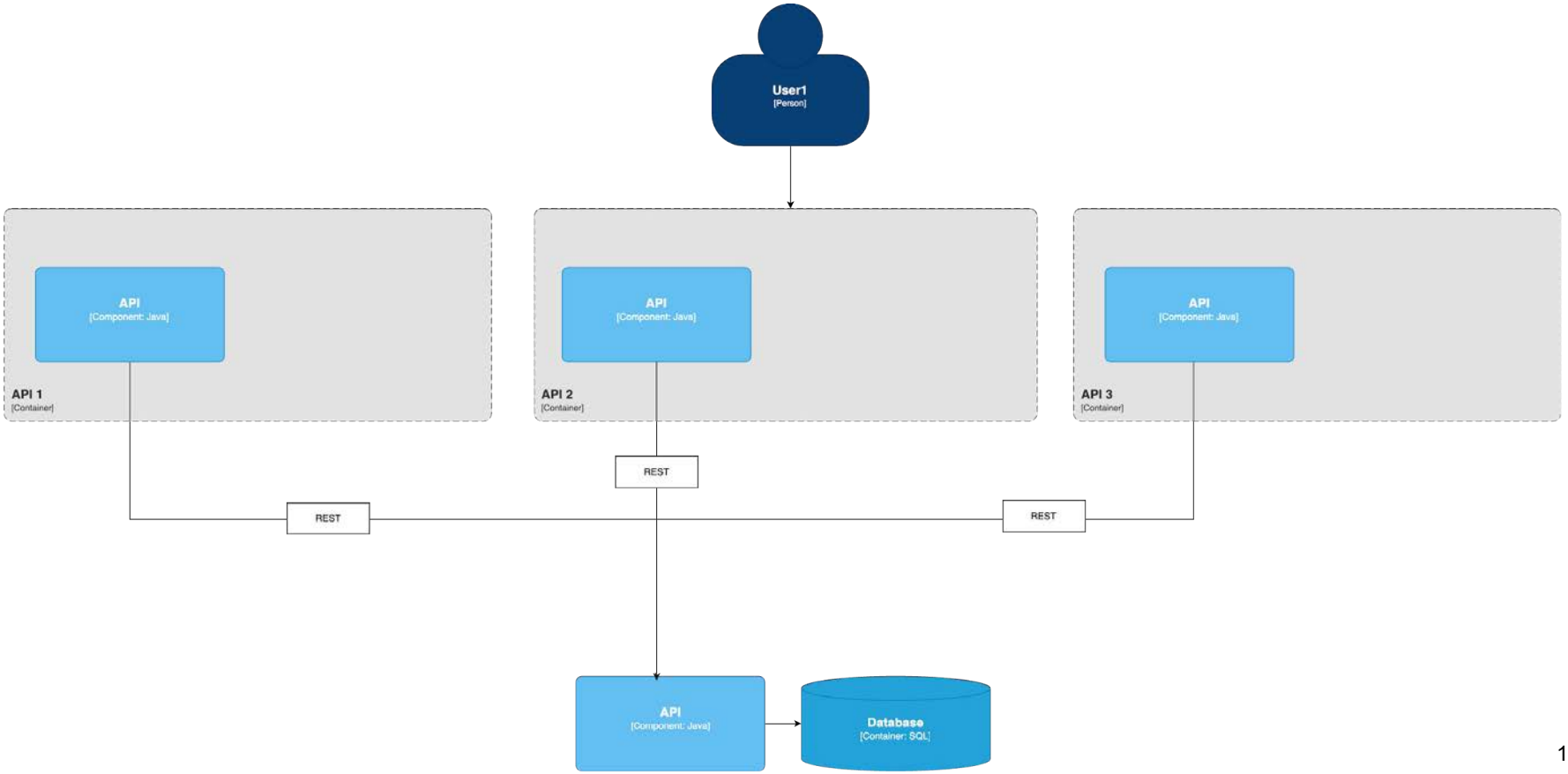
In-memory cache: Personalized caches



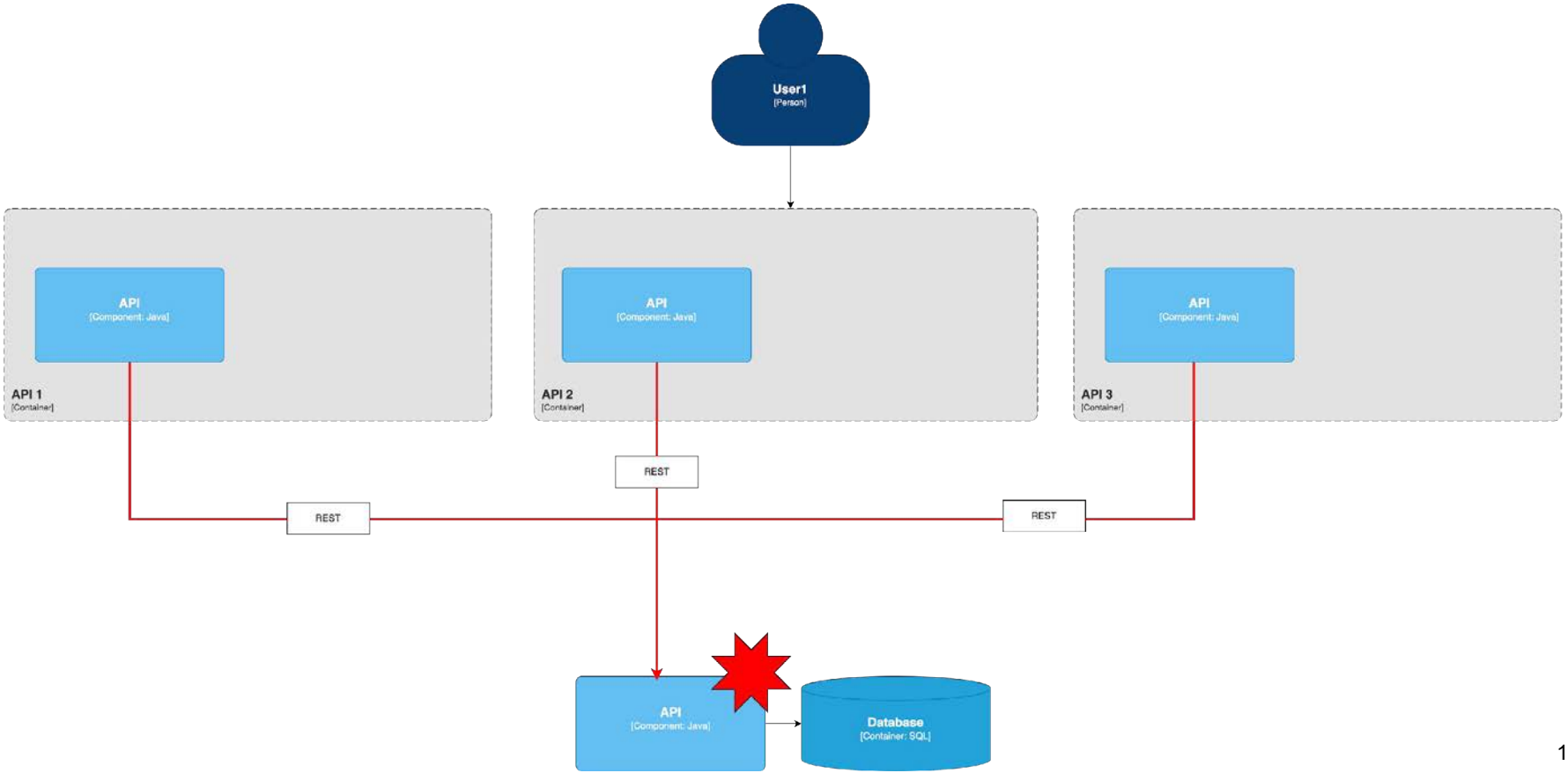
In-memory cache: Personalized caches



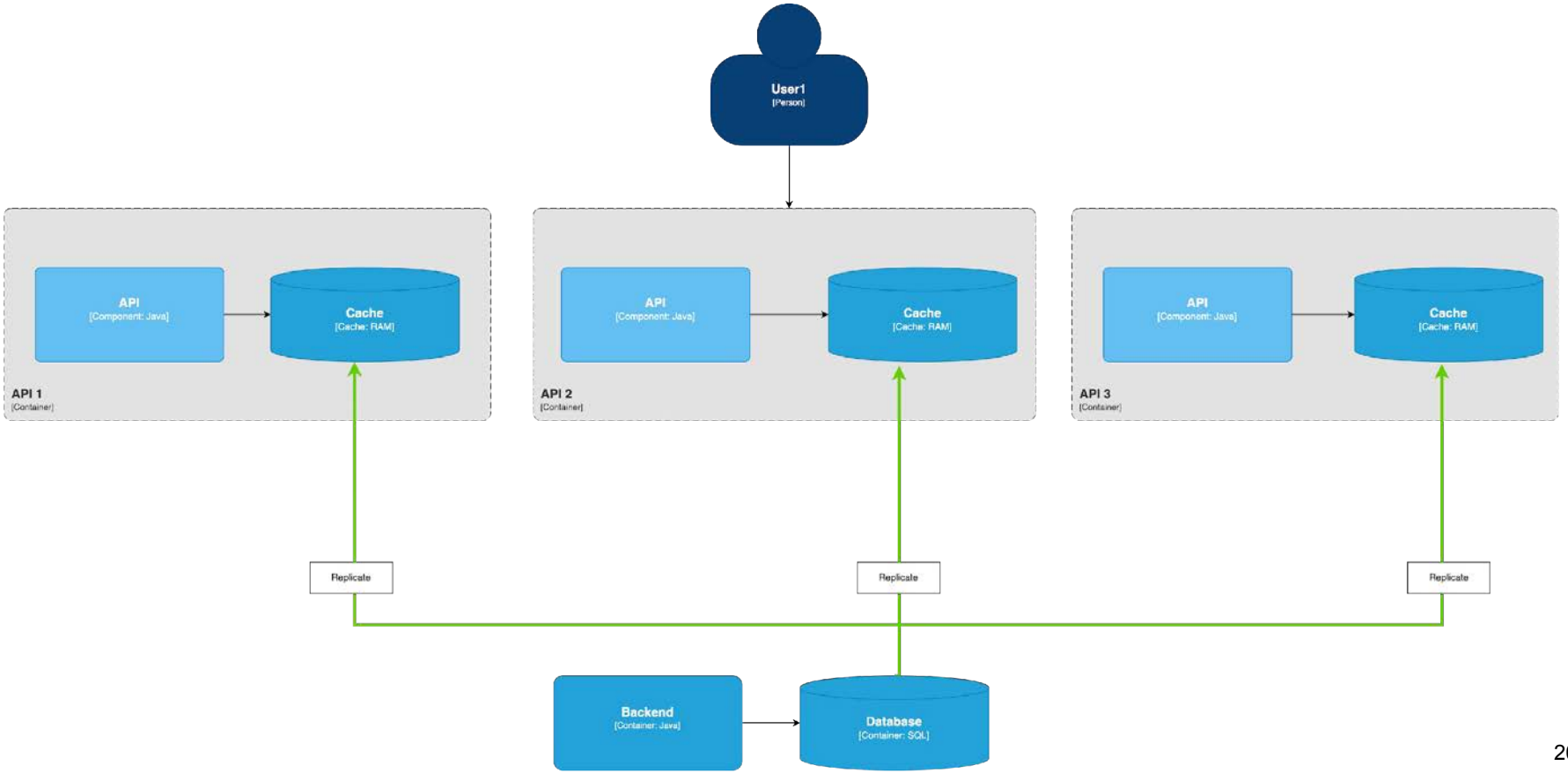
In-memory cache: local data replication



In-memory cache: local data replication



In-memory cache: local data replication



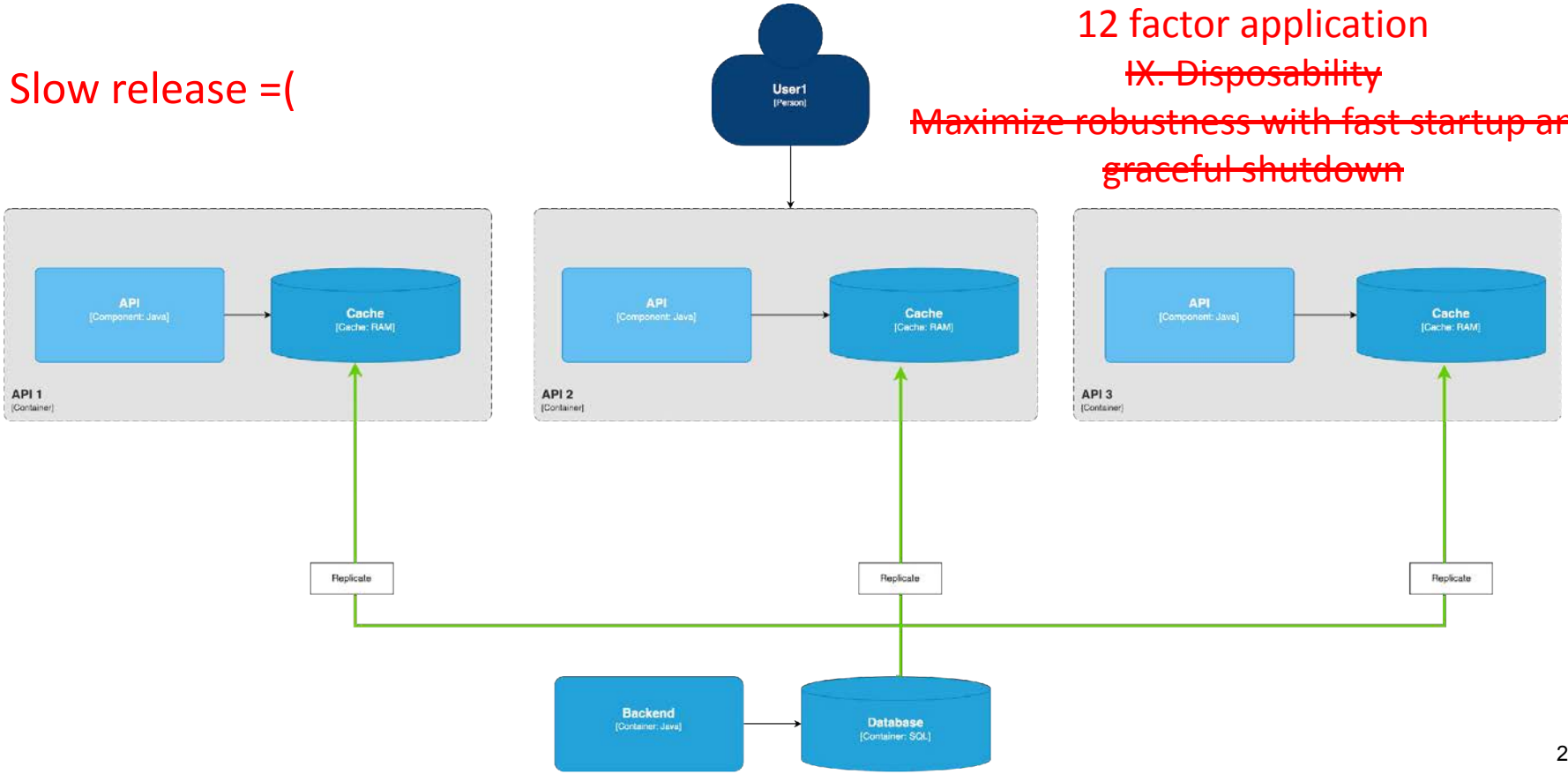
Inmemory cache: local data replication

Slow release =(

12 factor application

~~IX. Disposability~~

~~Maximize robustness with fast startup and graceful shutdown~~



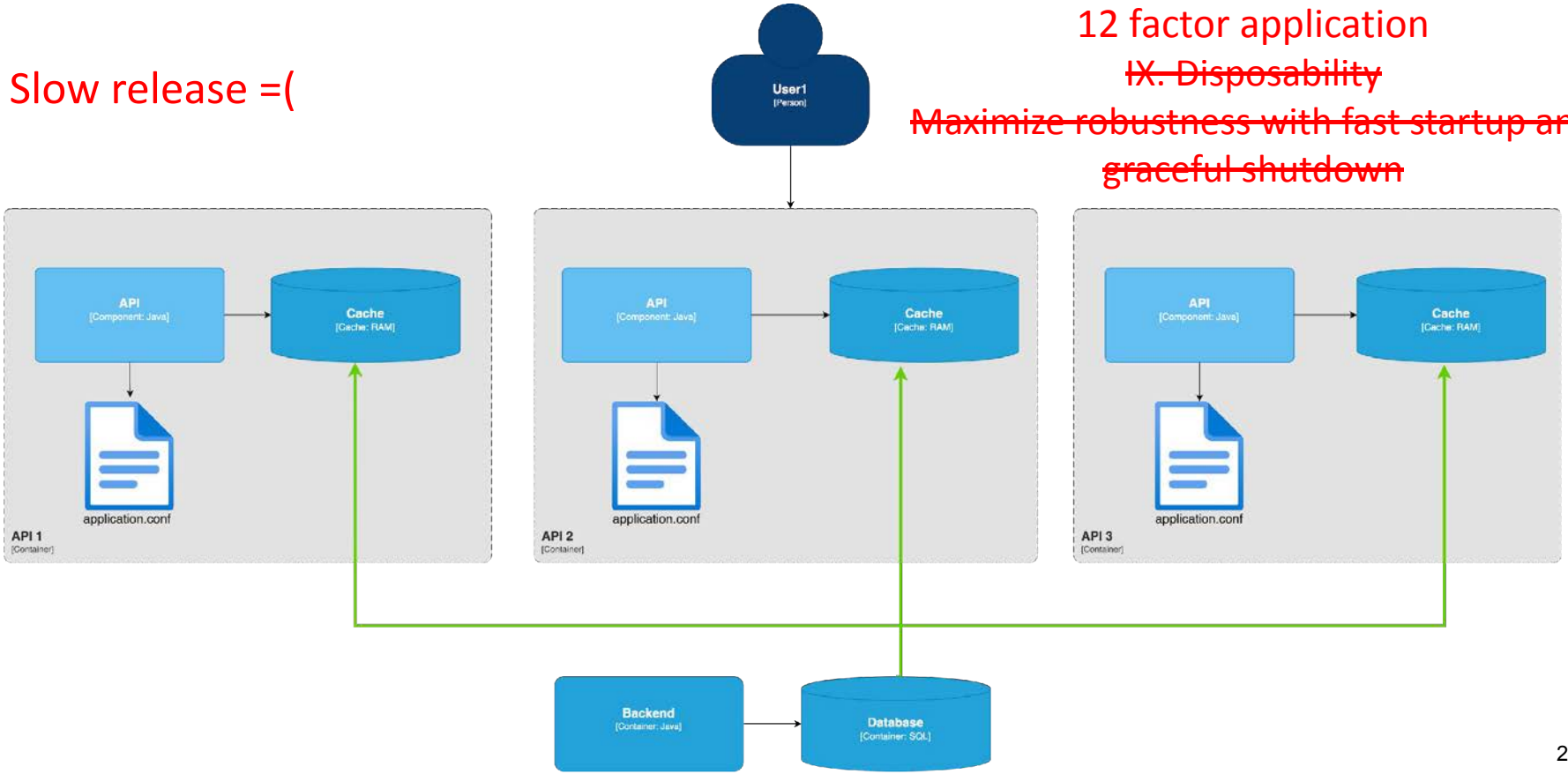
Inmemory cache: local data replication

Slow release =(

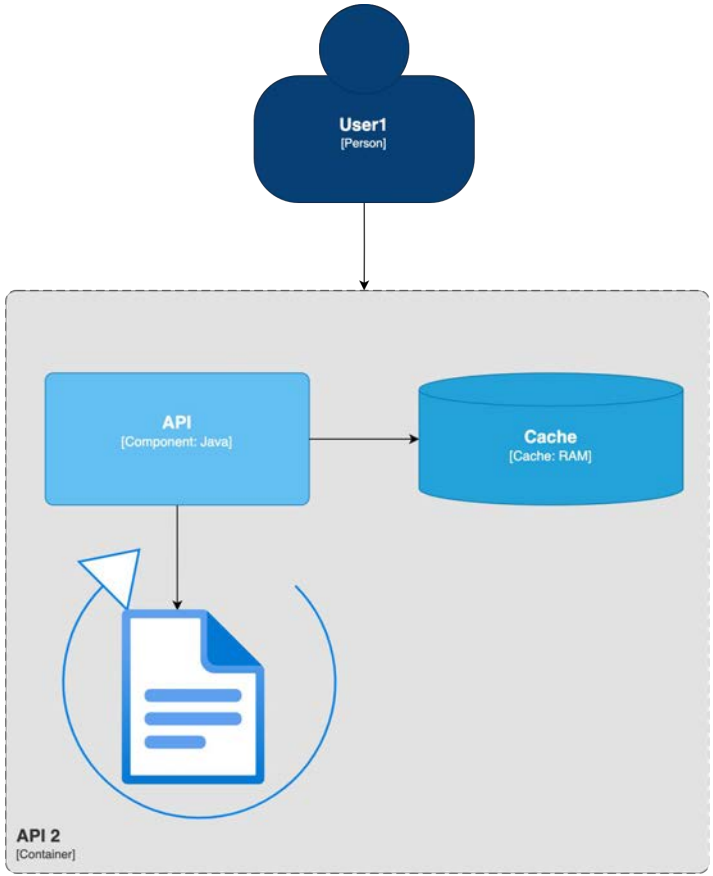
12 factor application

~~IX. Disposability~~

~~Maximize robustness with fast startup and graceful shutdown~~



Reloadable config



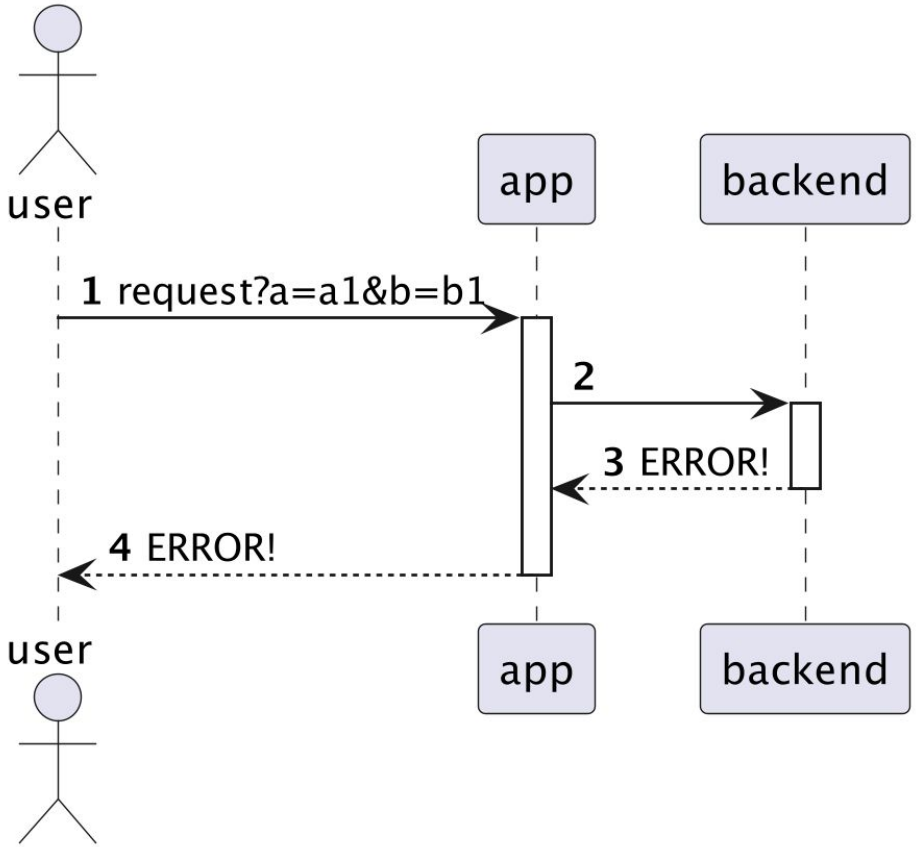
Reloadable config

```scala
class MyController1(
  config: MyConfig,
  myService: MyService,
) {
  def myMethod(request: MyRequest): Task[MyResponse] = {
    myService
      .doWork(request.user, config.maxConcurrent)
      .map(toViewModel(config.mappingMode))
  }
}
```

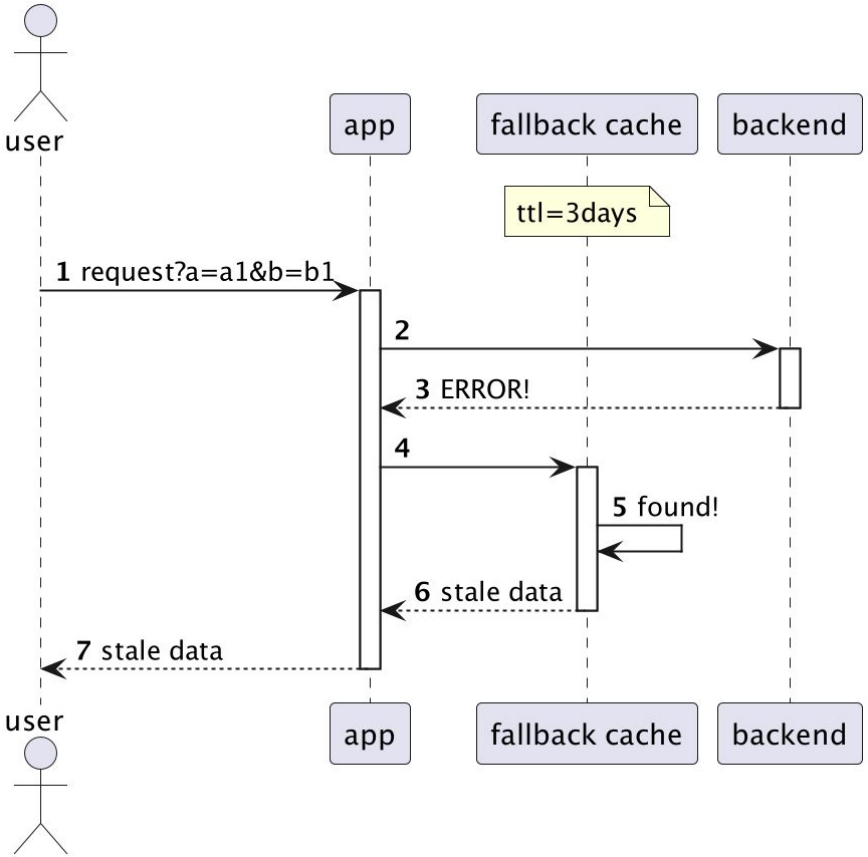
```scala
class MyController2(
  reloadableConfig: AtomicReference[MyConfig],
  myService: MyService,
) {
  def myMethod(request: MyRequest): Task[MyResponse] = {
    val config = reloadableConfig.get
    myService
      .doWork(request.user, config.maxConcurrent)
      .map(toViewModel(config.mappingMode))
  }
}
```

Fallback

Fallback: Problem

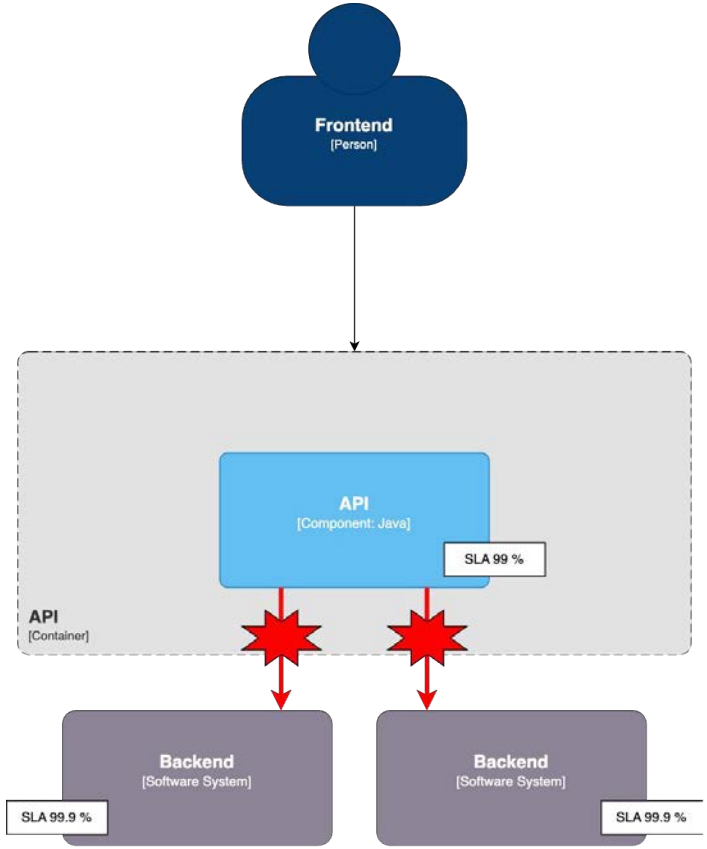


Fallback cache

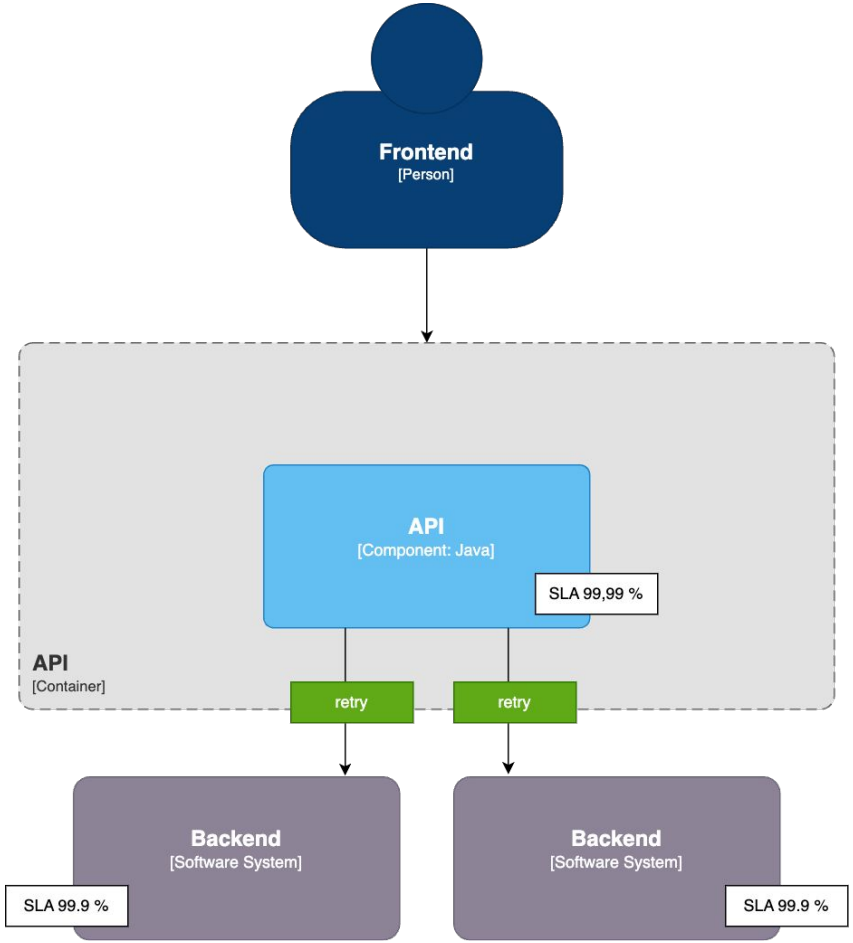


Retry

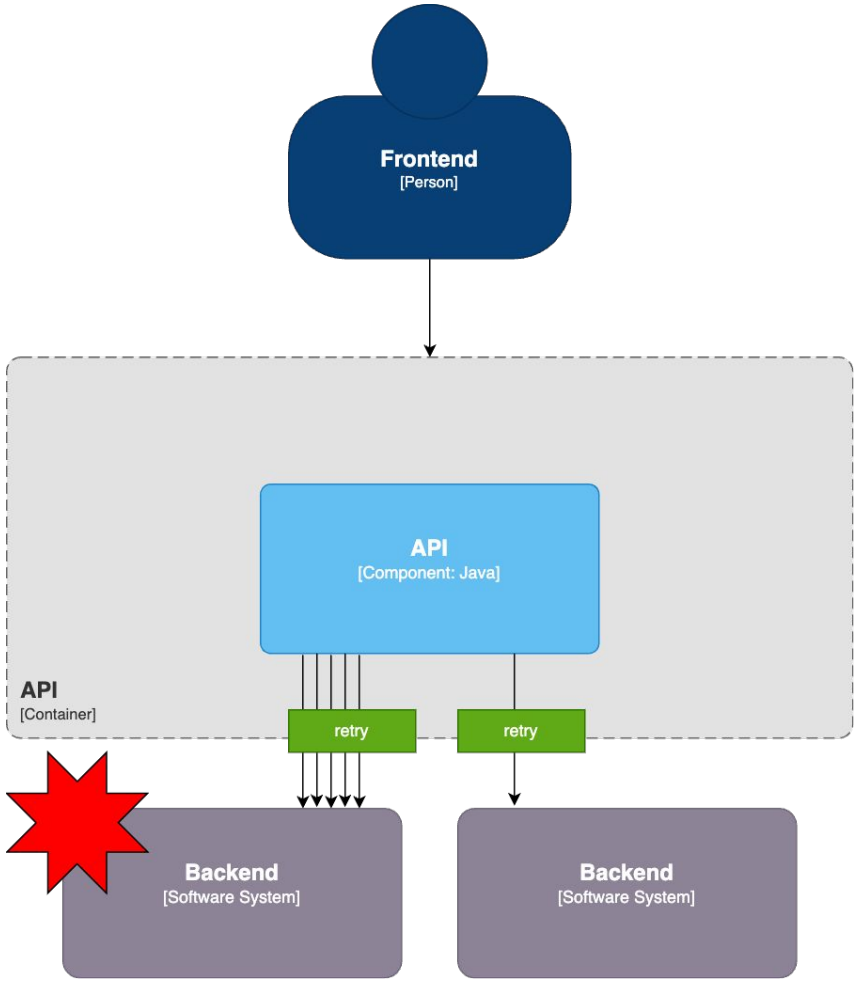
Retry


Frontend
[Person]
API
[Component: Java]
SLA 99 %
API
[Container]
Backend
[Software System]
SLA 99.9 %
Backend
[Software System]
SLA 99.9 %

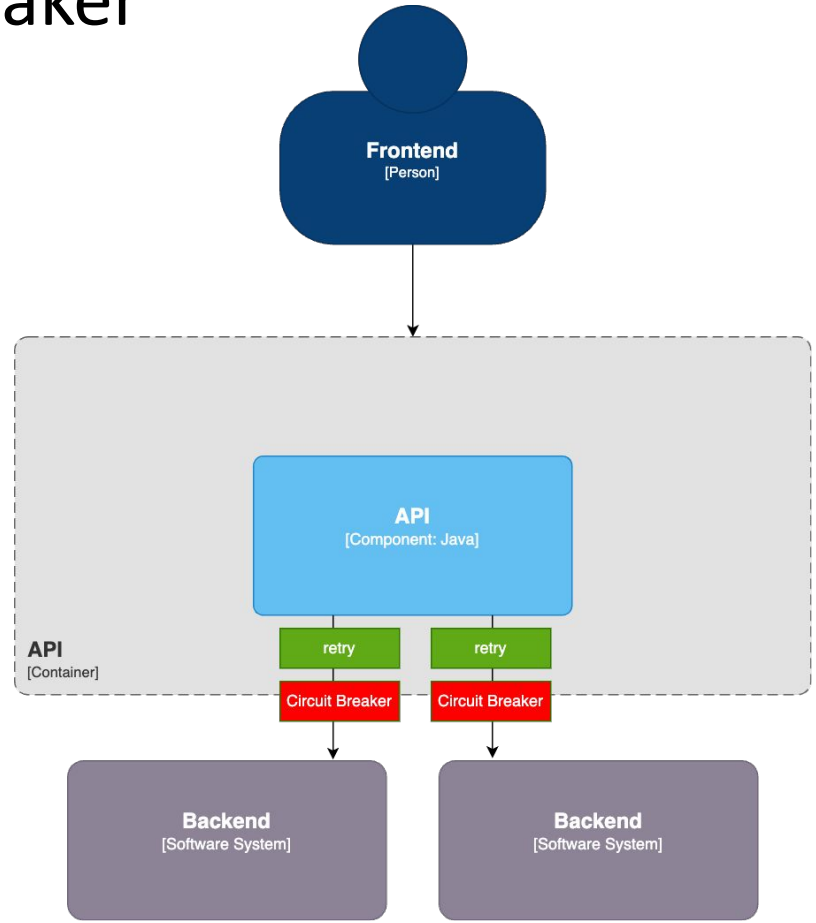
Retry

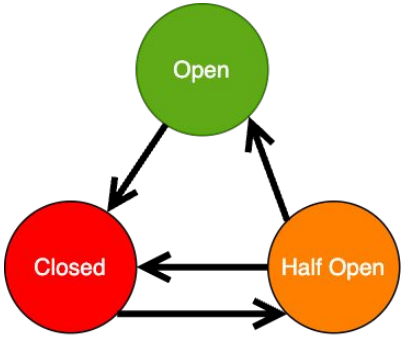

Frontend
[Person]
API
[Component: Java]
SLA 99,99 %
API
[Container]
retry
retry
Backend
[Software System]
SLA 99.9 %
Backend
[Software System]
SLA 99.9 %

Retry


Frontend
[Person]
API
[Component: Java]
API
[Container]
retry
retry
Backend
[Software System]
Backend
[Software System]

Circuit Breaker





Summarize

- Bulkhead
- Inmemory cache
- Reloadable config
- Fallback cache
- Retry
- Circuit Breaker

Fin

Dmitrii Pakhomov

Telegram: @ymatigoosa