LES TESTS DE PERFORMANCE

Speaker notes

authors:

- S.LAVAZAIS
- D.SIX

Sources:

- Wikipedia - software performance testing

INTRODUCTION - SOMMAIRE

1. Les tests performance, qu'est-ce que c'est ?
2. Quel type de test pratiquer?
3. Java Microbenchmark Harness
4. Taurus

zenika

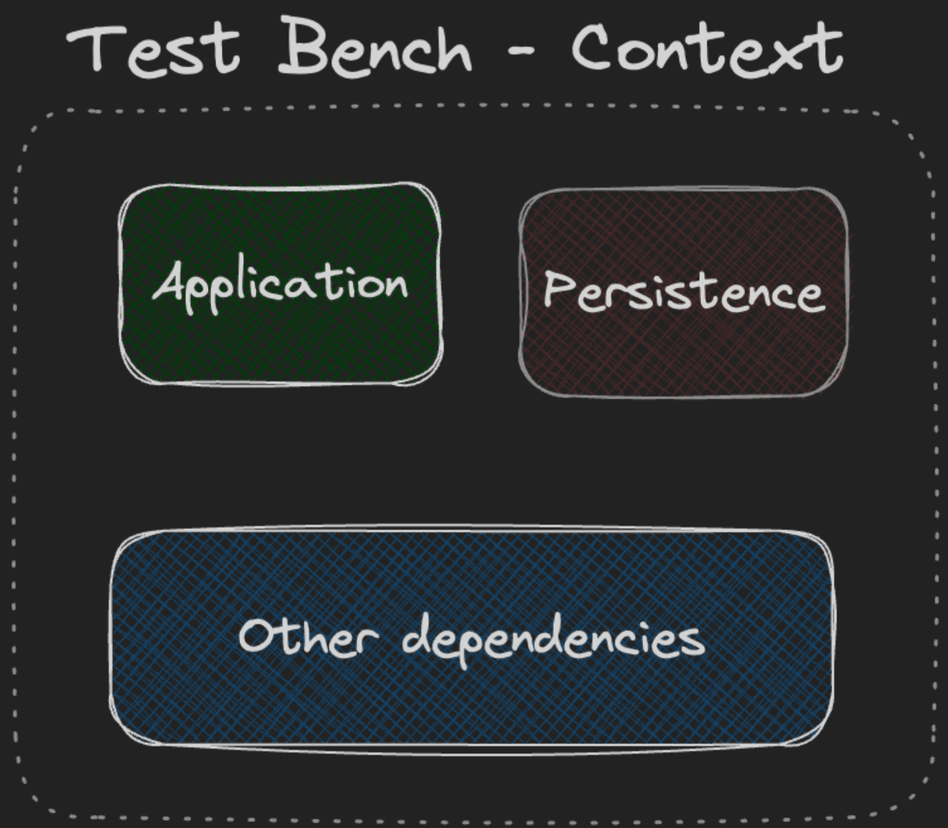
Speaker notes

For this presentation, we're going to see roughly through the performance testing, and we're going to see how to work a solution to make performance testing easier and automated, "Taurus" by Blazemeter.

But first, let me introduce you what is performance testing and how many kinds of tests we can practice against an application.

1. Les testes de performance, qu'est-ce que c'est ?
2. Quel type de test pratiquer?
    1. test en isolement
    2. test de charge
    3. test en stress
    4. test de pic
    5. test au limites
    6. test d'endurance
3. Tests en isolement avec Java Microbenchmark Harness
    1. Demo Isolement
4. Tests de charge avec Taurus
    1. Demo de charge
5. Merci !

LES TESTES PERFORMANCE, QU'EST-CE QUE C'EST ?



Speaker notes

Les tests de performances, c'est la capacité à mettre une application et toutes ses dépendances dans un context virtuel ou réel et observer comment ces composants ce comportent. On peut ainsi en déterminer des axes d'amélioration pour l'application testée.

(point à discuter avec @sixdouglas)

LES TESTES PERFORMANCE, QU'EST-CE QUE C'EST ?

temps de réponse

les resources utilisées

l'adaptation

zenika

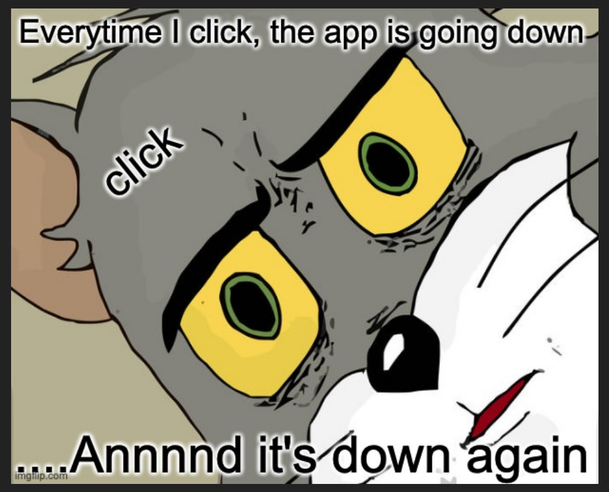
Speaker notes

Voici les principals métriques qu'un test de performance évalue :

- Le temps de réponse global (sur le pourcentage de toute les réponses) et à un instant T.
- La quantité de resources utilisées
- Mesure de la capacité d'adaptation / de la scalabilité

LES TESTES PERFORMANCE, QU'EST-CE QUE C'EST ?

les coupûres



bug / bottleneck

la sécurité

zenika

2 . 3

Speaker notes

Les tests de performance permet également de :

- De prévenir les coupures dû à la charge
- De détecter les bugs non couverts / les goulots d'étranglement
- D'atténuer les failles de sécurités

QUEL TYPE DE TEST PRATIQUER?

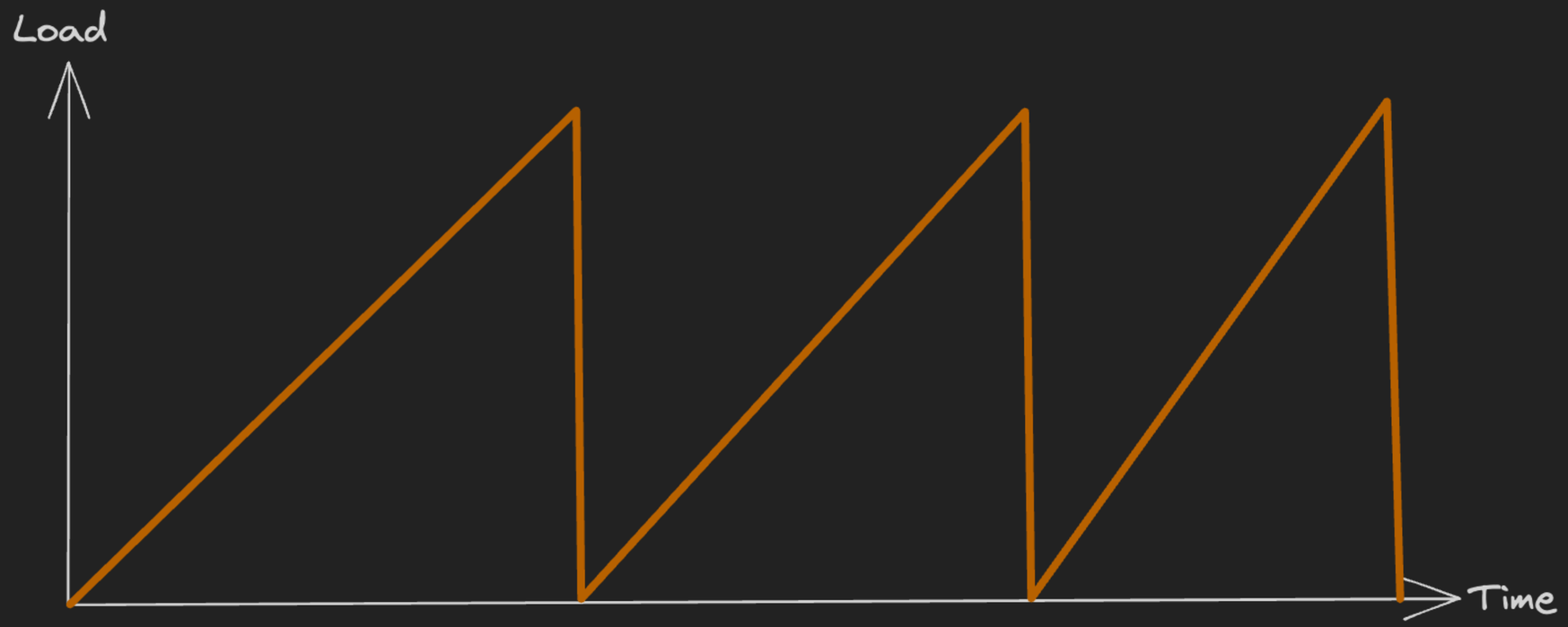
1. test en isolement
2. test de charge
3. test en stress
4. test de pic
5. test aux limites
6. test d'endurance

zenika

Speaker notes

Voici les différents tests de performance praticable ainsi que les diférences entre eux.

TEST EN ISOLEMENT

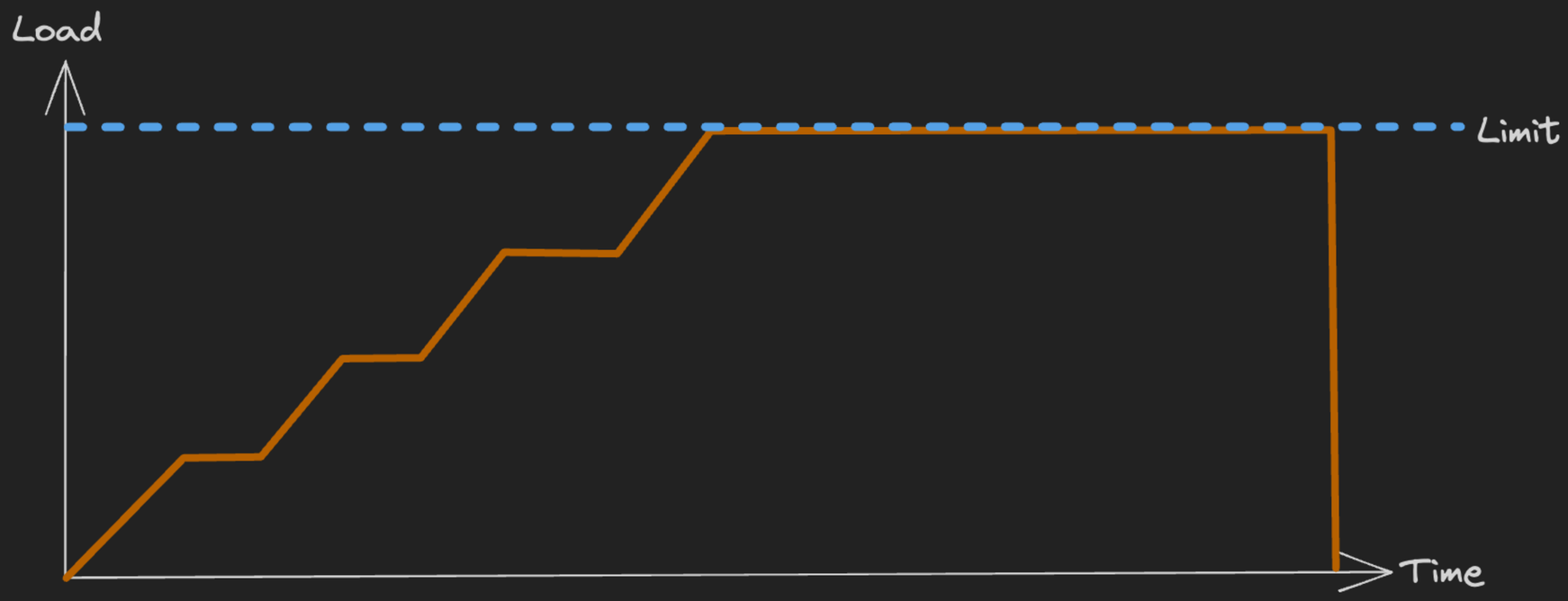


zenika

Speaker notes

Ce type de test est réalisé sur un *banc de test* (pour un cas isolé) en pratiquant une répétition d'exécutions.

TEST DE CHARGE



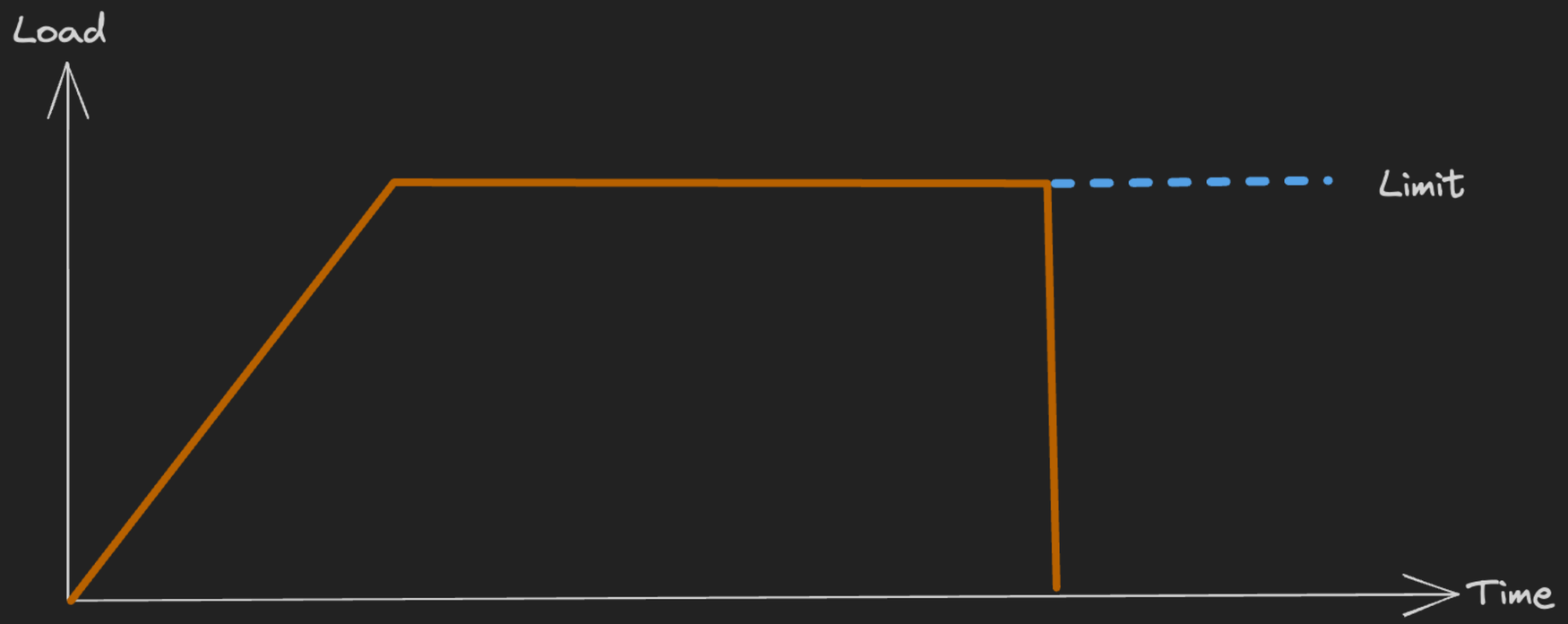
zenika

Speaker notes

Le test de charge est la forme la plus simple pour tester une application. L'objectif est de vérifier si l'application est capable de gérer les limites de temps de réponse / consommation ressources qui ont été décider préalablement (par exemple au travers d'un SLA)
L'infrastructure est également sous-monitoring durant cette phase de test.

Ce type de test peut être utilisé comme un test de qualité d'une release à une autre, ainsi qu'un objectif à maintenir.

TEST EN STRESS

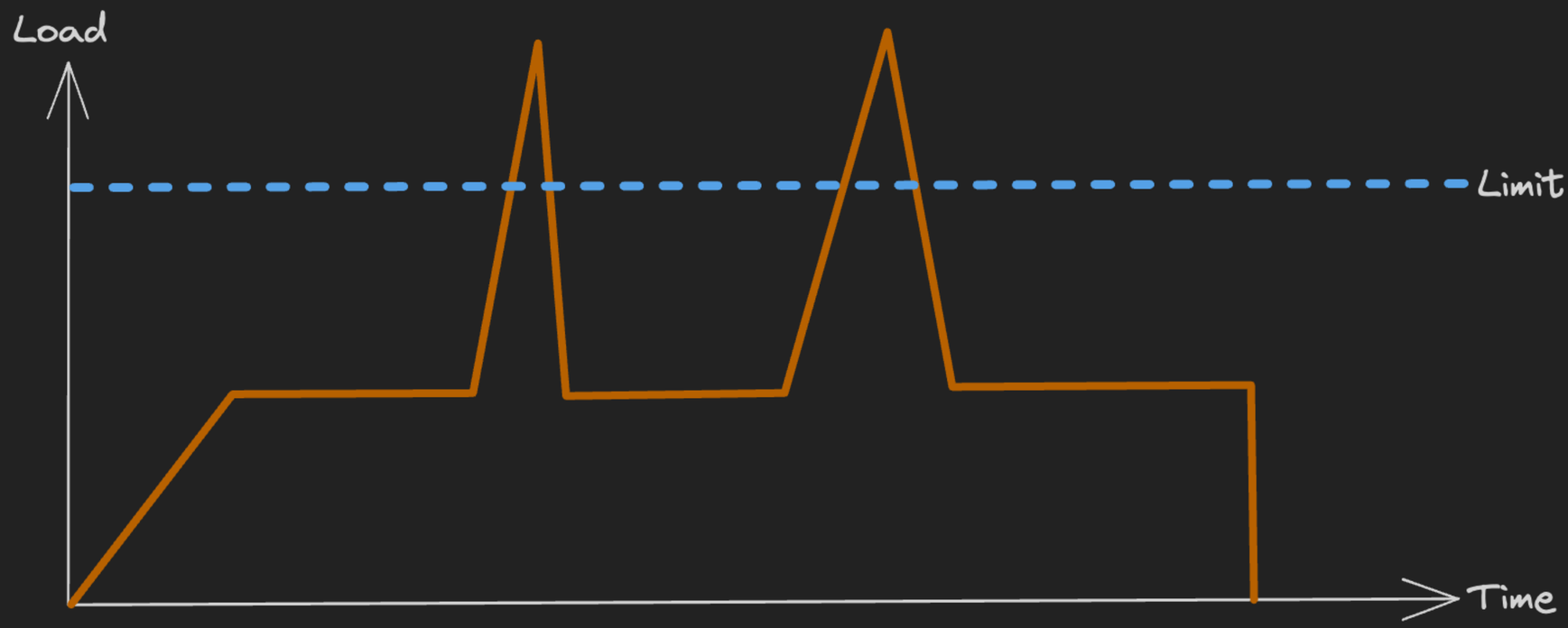


zenika

Speaker notes

Ce type de test vise principalement à déterminer les limites hautes de l'application et/ou de ses dépendances.

TEST DE PIC

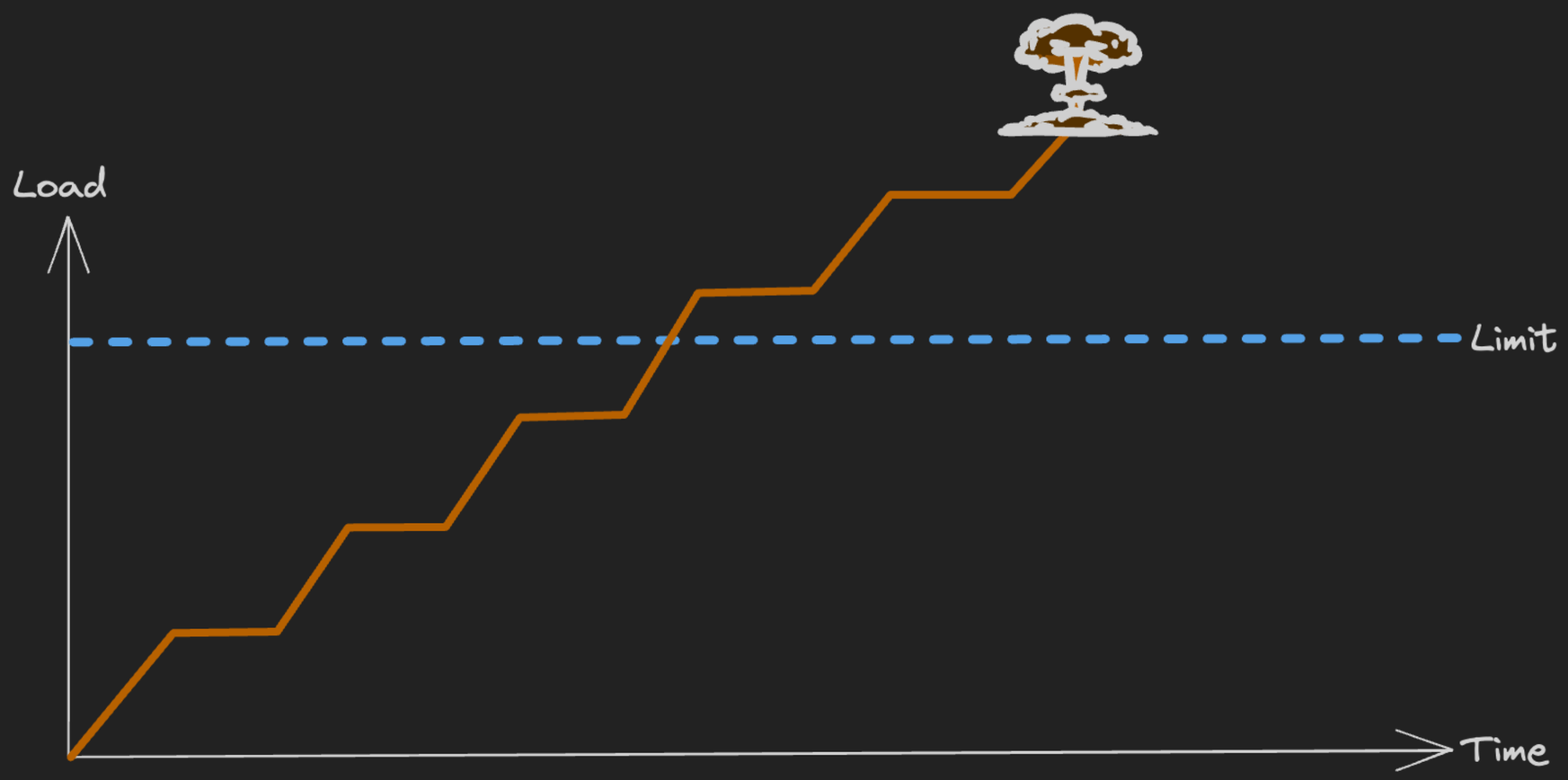


zenika

Speaker notes

Le test de pic vise à déterminer les problèmes de performance quand un changement de contexte se produit sur l'application testée, que ce soit une montée en charge soudaine du nombre d'utilisateurs qui se connecte en même temps ou à l'inverse une baisse de charge.

TEST AUX LIMITES

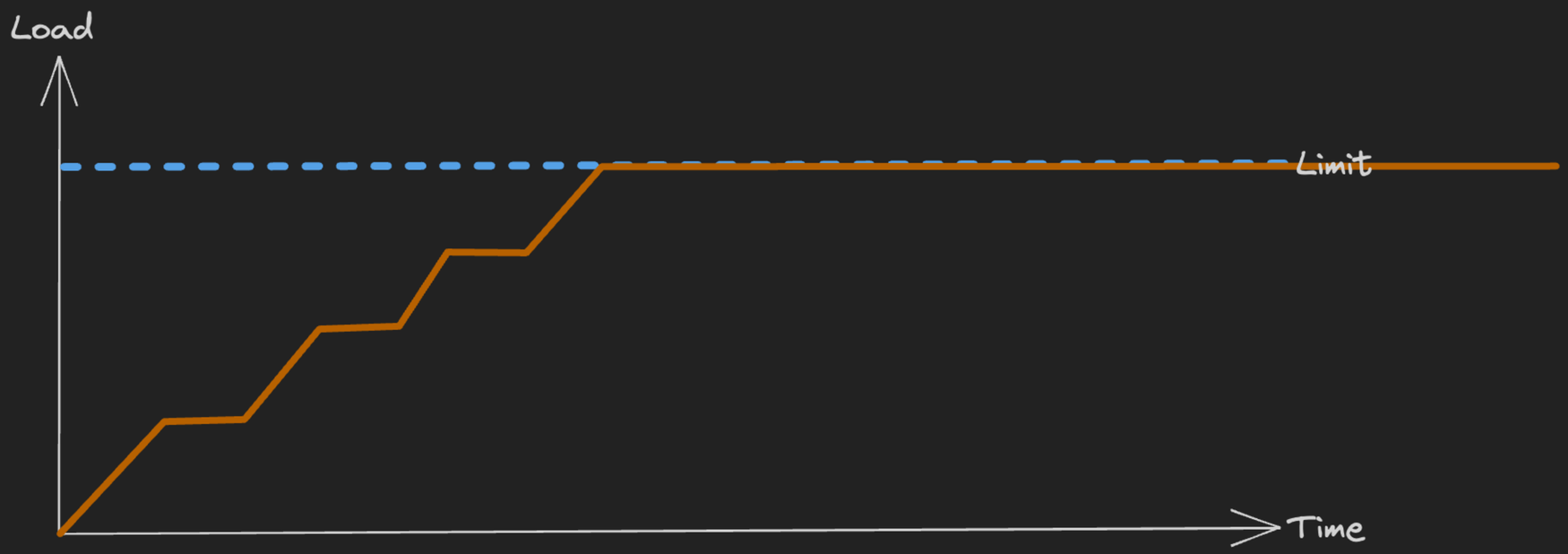


zenika

Speaker notes

Le test au limites permet d'évaluer le point de rupture de l'application. On parle souvent de test de capacité car il est utile pour déterminer si le SLA est judicieux pour l'application testée.

TEST D'ENDURANCE



zenika

Speaker notes

Le test d'endurance consiste à maintenir une charge pendant un temps extrêmement long et permet déterminer si l'application testée est capable de supporter un tel scénario. Comme les tests de charge l'infrastructure est également sous-monitoring durant cette phase de test.

MICROBENCHMARK

Outils utilisé: Open JDK JMH

Speaker notes

JMH est un outil proposé et maintenu par OpenJDK

CONCEPTS

Choix du type d'analyse

Définition d'un `State`

Écriture du code à tester

Écriture du `Benchmark`

zenika

Speaker notes

No notes on this slide.

TYPES D'ANALYSES

- Throughput: ops/time
- AverageTime: time/op
- SampleTime: sampling exec time
- SingleShotTime: exec once

Speaker notes

- Throughput: pour un temps donné, compte le nombre d'exec de la fonction
- AverageTime: pour un nombre d'exécution donné, mesure le temps
- SampleTime: exécute la fonction en continue et échantillon de temps
- SingleShotTime: exécute une fois la fonction, idéal pour exec à froid

STATE

```java
@State(Scope.Benchmark)
public static class ExecutionPlan {
    public List<Brand> brandList = new ArrayList<>(10000);

    @Setup
    public void setup() throws IOException {
        for (int i = 0; i < 10000; i++) {
            Brand builtBrand = buildBrand();
            brandList.add(builtBrand);
        }
    }
}
```

Speaker notes

- Petite annotation qui va bien
- Une petite liste dans laquelle nous allons chercher
- Une fonction pour remplir notre liste

CODE À TESTER

```java
private Brand findBrandStream(List<Brand> brandList, Integer id) {
    return brandList.stream().filter(brand -> brand.id().equals(id))
            .findFirst()
            .orElse(null);
}

private Brand findBrandFor(List<Brand> brandList, Integer id) {
    for (Brand brand : brandList) {
        if (brand.id().equals(id)) {
            return brand;
        }
    }
    return null;
}
```

Speaker notes

- Une fonction qui utilise une `stream` pour chercher par ID dans la `List`
- Une fonction qui utilise une boucle `ForEach` pour chercher par ID dans la `List`

DÉFINITION DU BENCHMARK 1/2

```java
@Benchmark
@OperationsPerInvocation(10000)
public void findBrandStream(Blackhole blackhole, ExecutionPlan plan) {
    for (int i = 0; i < 10000; i++) {
      blackhole.consume(
        findBrandStream(
          plan.brandList,
          plan.brandList.get(random.nextInt(10000)).id()
        )
      );
    }
}
```

zenika

Speaker notes

No notes on this slide.

DÉFINITION DU BENCHMARK 2/2

```java
@Benchmark
@OperationsPerInvocation(10000)
public void findBrandFor(Blackhole blackhole, ExecutionPlan plan) {
    for (int i = 0; i < 10000; i++) {
      blackhole.consume(
        findBrandFor(
          plan.brandList,
          plan.brandList.get(random.nextInt(10000)).id()
        )
      );
    }
}
```


Speaker notes

No notes on this slide.

EXÉCUTION

```
# Blackhole mode: compiler (auto-detected, use -Djmh.blackhole.autoDetect=false
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: org.example.StreamVsForMain.findBrandFor

# Run progress: 0,00% complete, ETA 00:03:20
# Fork: 1 of 1
# Warmup Iteration   1: 76239,113 ops/s
# Warmup Iteration   2: 76767,384 ops/s

Iteration   1: 76972,113 ops/s
Iteration   2: 76562,211 ops/s
```

zenika

Speaker notes

No notes on this slide.

RÉSULTATS

```
Benchmark                              Mode  Cnt      Score      Error  Units
StreamVsForMain.findBrandFor          thrpt    5  76758,517 ±  559,708  ops/s
StreamVsForMain.findBrandStream       thrpt    5  43736,684 ± 1081,065  ops/s
```


Speaker notes

No notes on this slide.

TESTS DE CHARGE AVEC TAURUS

zenika

Speaker notes

Taurus est un kit de développement qui permet à la fois :

- De pouvoir exécuter des tests avec plusieurs frameworks différents de test (sans qu'il y est une grande différence dans l'implémentation du test)
- De pouvoir exécuter des tests aussi bien en local que sur un cloud provider. (plateforme Blazemeter)

TESTS DE CHARGE AVEC TAURUS

zenika

Speaker notes

Les tests de charge avec Taurus sont assez simple, ils sont constitué de deux parties :

- la configuration de l'exécution
- la configuration du/des scénario(s)

TESTS DE CHARGE AVEC TAURUS

zenika

Speaker notes

Pour notre petite démonstration, on va prendre l'exemple d'une API qui gère la déposition de données en amont d'un système d'information

TESTS DE CHARGE AVEC TAURUS

zenika

Speaker notes

Tout d'abords les paramètres d'exécution qui gère donc la volumétrie et le temps d'exécution du scenario de test:

- 100 utilisateurs
- 5000 requêtes par secondes (total reparti sur les 100 utilisateurs)
- sur 1 heure

TESTS DE CHARGE AVEC TAURUS

zenika

Speaker notes

Pour tester notre API, on va donc avoir un scénario assez simple qui va simuler:

1. un enregistrement d'une donnée
2. une lecture de ce nouvel enregistrement
3. une modification de ce nouvel enregistrement

Ce scénario sera exécuté par tous les utilisateurs virtuels