手動かして学ぶ 負荷試験

サポーターズCoLab勉強会 @ 渋谷スクエア

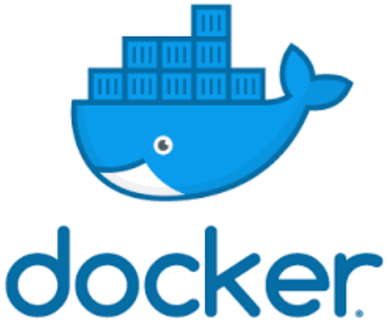
2018/07/23

サポーターズ CoLab

Contents

はじめに: 環境の用意

今日使うもの



For Mac

https://docs.docker.com/v17.12/docker-for-mac/install/

For Windows 10

https://docs.docker.com/docker-for-windows/install/

For 他の人

GCPかAWSを使ってインスタンスを作る
(後で記述する; GCPを使う)

コマンドライン操作
が可能なターミナル

上記のインストールは
ネットワークの負荷をかけるので
USBで installerを配布します

GCP

ブラウザから 全部できる & 無料トライアルがあるので、良さそう

ブラウザはChromeの方が良いかも？

https://cloud.google.com/free/?hl=ja

実際にポチポチぽち

https://cloud.google.com/free/?hl=ja

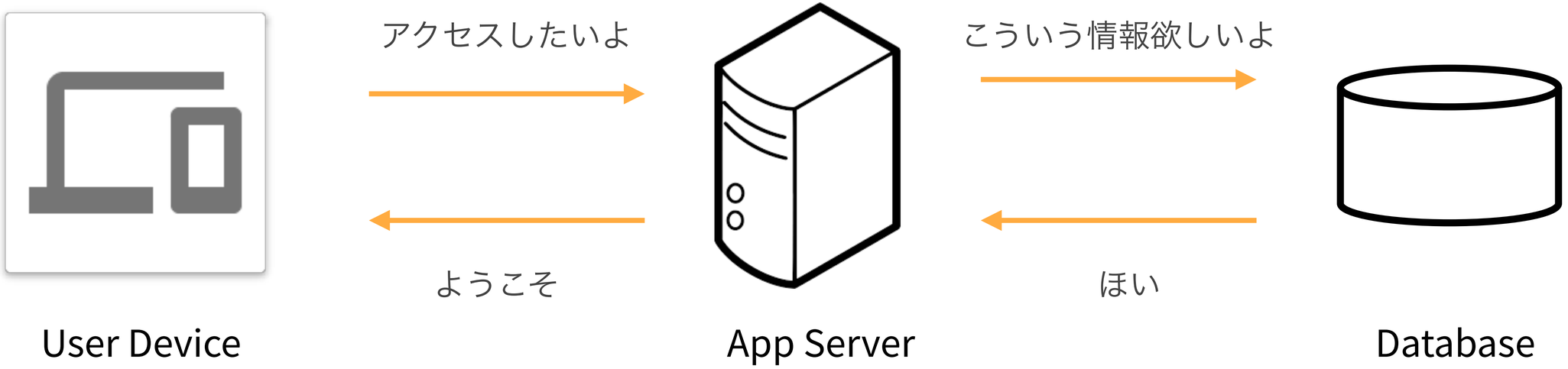
ここの僕一人の手で足りないので
できた人から近くの人に教えてもらえると助かります

ここで時間を潰すのもあれなので
15分ぐらいでできない場合は、一旦諦めます

Webサーバーを立ち上げてみる (ハンズオン)
golang と python サーバそれぞれ

https://github.com/TKNGUE/supporterz-colab-20181122

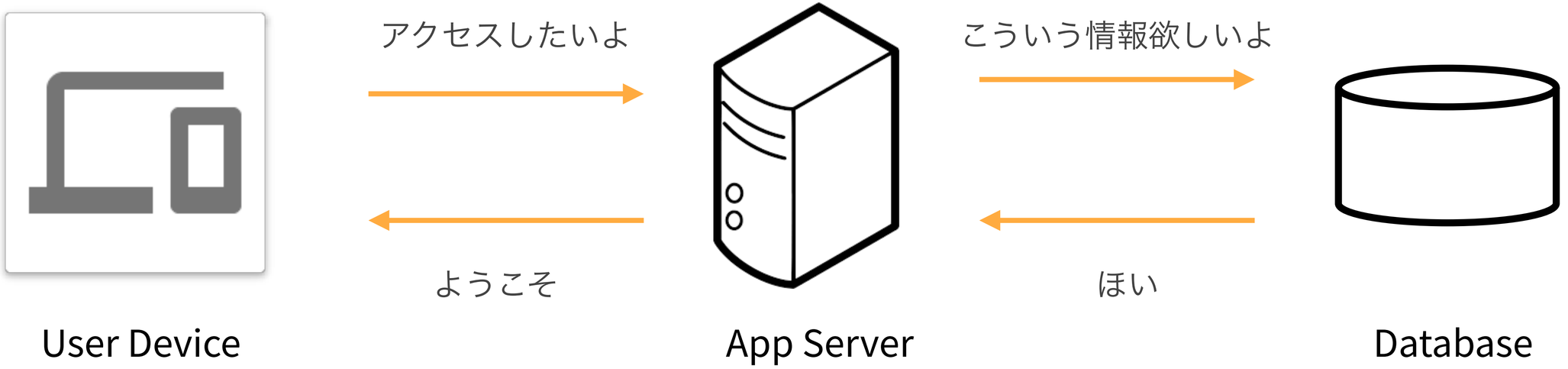
サービスってどう動いているの



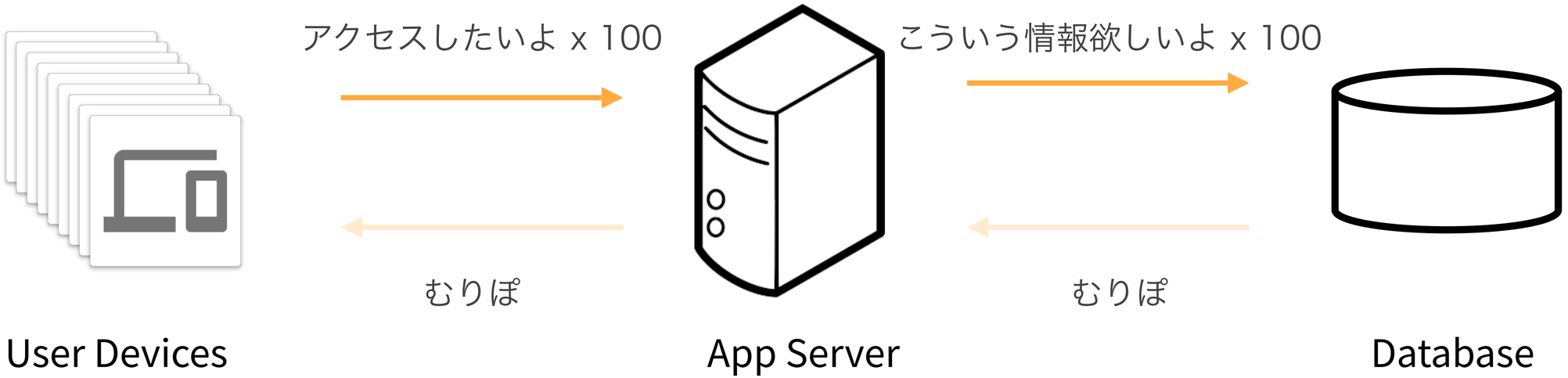
※最低限の構成: 大きくなると構成がもっと複雑になります

負荷試験とは

負荷試験： サービスに実際に負荷をかけて試験してみること



負荷試験： サービスに実際に負荷をかけて試験してみること



構成したサービスがどれぐらいの負荷に耐えられるか？
あるサービスに負荷がかかったときに全体としてどんな挙動になるのか？
* Databaseに負荷がかかるとどうなるのか？
* App Serverに負荷がかかるとどうなるのか？

負荷をかけるためのツールはたくさんある

the Grinder

Gatling

Tsung

Locust

JMeter

wrk

apache-bench (ab)

siege

負荷をかけるためのツールはたくさんある

the Grinder

Gatling

Tsung

Locust

JMeter

wrk

apache-bench (ab)

siege

どれつかえばいいんだ…

負荷をかけるためのツールはたくさんある

グラフが綺麗なやつ？

複雑なシナリオができる？

お手軽にできるやつ？

負荷をかけるためのツールはたくさんある

グラフが綺麗なやつ？

複雑なシナリオができる？

お手軽にできるやつ？

で選ぶ前に！

間違った計測をしないための知識をおさえておく！

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

→ リクエストを送って返ってきた時間で計測する

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

→ リクエストを送って返ってきた時間で計測する

1回のリクエストだけで大丈夫？

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

→ リクエストを送って返ってきた時間で計測する

1回のリクエストだけで大丈夫？

→ 複数回の結果を平均する

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

→ リクエストを送って返ってきた時間で計測する

1回のリクエストだけで大丈夫？

→ 複数回の結果を平均する

1回目: 123ms
2回目: 124ms　の場合
3回目: 122ms

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

→ リクエストを送って返ってきた時間で計測する

1回のリクエストだけで大丈夫？

→ 複数回の結果を平均する

1回目: 123ms
2回目: 124ms　の場合
3回目: 122ms

→ (123ms+ 124ms + 122ms) / 3 = 122ms !!

今回はサーバの応答時間を測ることを考える…

サーバの応答時間を測るには
どうしたら良いだろうか？

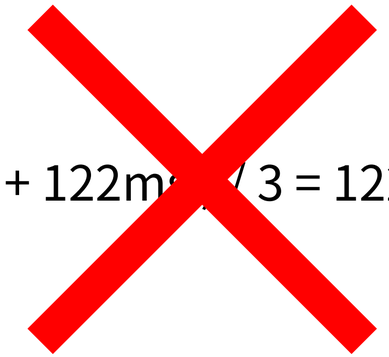
→ リクエストを送って返ってきた時間で計測する

1回のリクエストだけで大丈夫？

→ 複数回の結果を平均する

1回目: 123ms
2回目: 124ms　の場合
3回目: 122ms

→ (123ms+ 124ms + 122ms) / 3 = 122ms !!



今回はサーバの応答時間を測ることを考える…

1回目: 123ms
2回目: 124ms　　の場合どうなる？　　→ (123ms+ 124ms + 122ms + 1220ms) / 4 = 397ms !!
3回目: 122ms
4回目: 1220ms

1回目: 123ms
2回目: 124ms　　の場合どうなる？　　→ 3050000 ms !!
3回目: 122ms
4回目: 12200000ms

(相加)平均値ではなくて、中央値(50%位)を見よう！

今回はサーバの応答時間を測ることを考える…

50%位だけで十分か？

どっちの方が良いか？

- 50%位(中央値)で 100ms で 90%位で10秒かかるシステム
- **50%位(中央値)で 200ms で 90%位で1秒かかるシステム**

理想は 50%位, 90%位, 99%位, 99.9%位 …で見れると良い！

点で捉えるのではなく、面で応答時間を捉えよう！

ここまでを実際にやってみる

https://github.com/William-Yeh/docker-wrk

特定のサービスに負荷をかけないこと = 犯罪になります

これで計測できるように…!

まだ不十分…

計測器の処理も正しく理解する必要がある

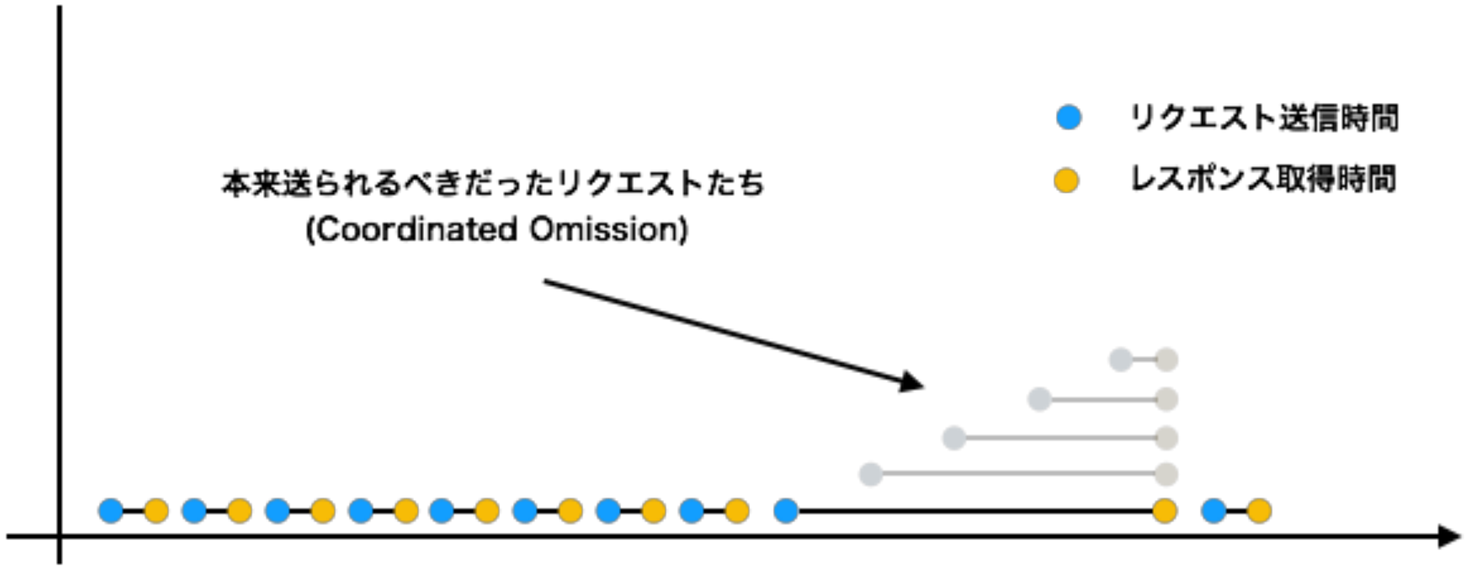
→ Coordinated Omission について紹介

Coordinated Omission (以下CO) とは…？

負荷試験の際に起こる問題で **実際の性能よりもサーバの性能を良く見せる計測誤差**

負荷試験中にサーバが停止などによって処理が遅延したときに、

計測器側がその遅延の評価を低く見積もりすぎてしまう誤差



Coordinated Omission (以下CO) とは…？

スレッド数1で秒間100リクエストの負荷試験を10秒間行なった場合を考えます

通常は1msで返すサーバが、どこかで1sだけ停止し、反応に1sかかったとします

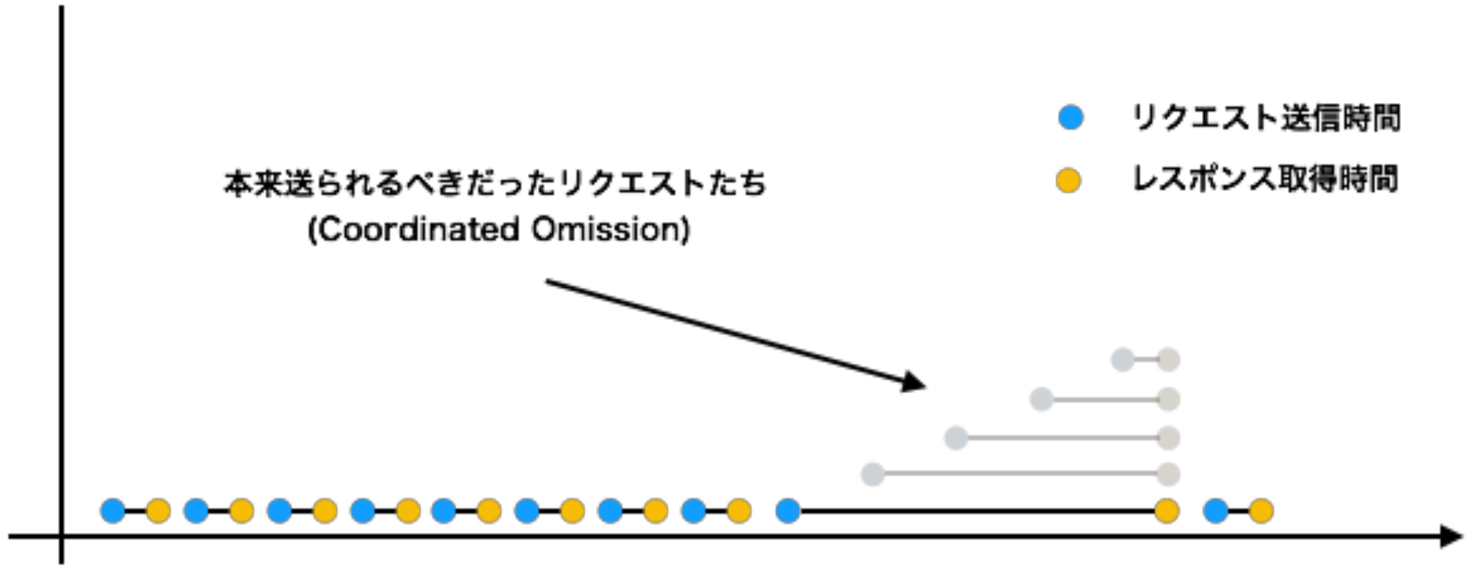
$$t_r = (9 \cdot 100[req./s] \cdot 1[ms] + 1 \cdot 1[req./s] \cdot 1[s])/(900 + 1) = 2[ms/req.]$$

$$t_r = (900 + 50500)/1000 = 514[ms/req.]$$

正しくは 1sかかっている間にも0.01sごとにリクエストが送られるはずなので

$$1000[ms] + 990[ms] + 980[ms] + \ldots + 10[ms] = 100 \cdot (10 + 1000)/2 = 50500$$

250倍のズレ！！！



ここまでを実際にやってみる

特定のサービスに負荷をかけないこと = 犯罪になります

おわりに

負荷試験とはの説明から実際にやってみました

dockerを使えば負荷試験の環境も簡単に用意できます

また計測するためには、ツールを使えば良いだけではなく
正しい計測の知識を付けることも同様に大事です

データベースに接続されたサービスを自分で作ってみて
実際に負荷試験をかけたりして遊んでみましょう