CLD Google App Engine PaaS

Auteurs: Edwin Haeffner, Arthur Junod Group: L4GrE Date: 10 May, 2024

Task 1: Deployment of a simple web application

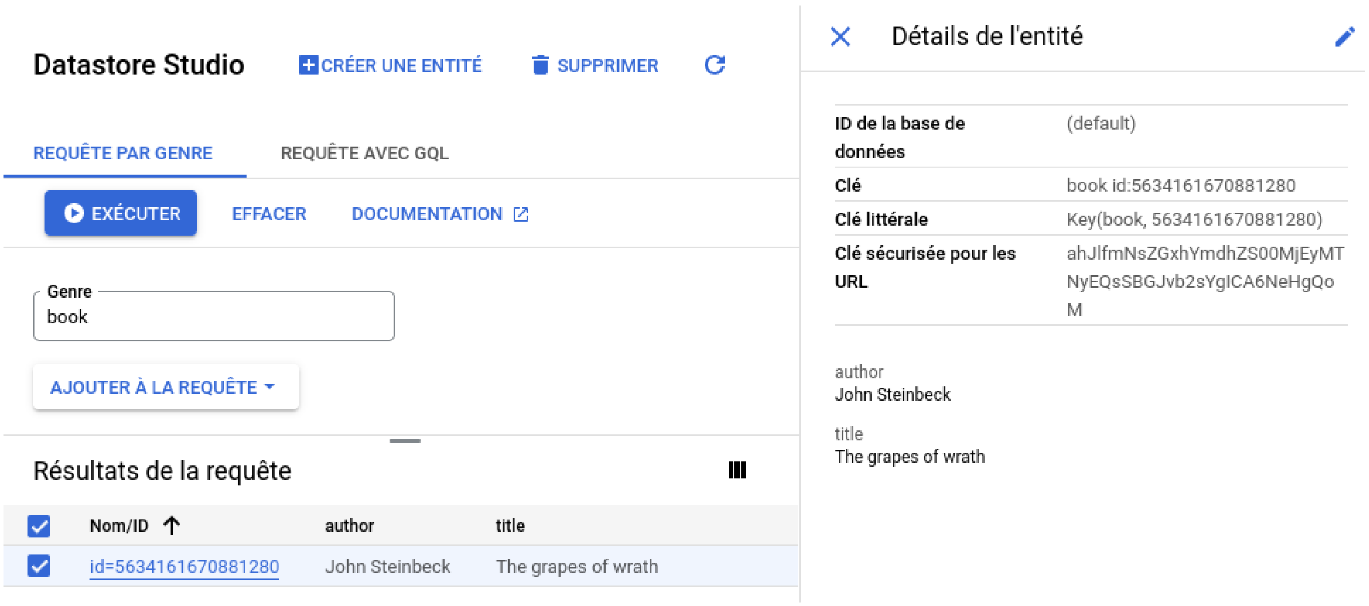
> Copy the Maven command to the report.

Maven command:

```
./mvnw clean package --batch-mode -DskipTests -Dhttp.keepAlive=false -
f=pom.xml --quiet
```

Task 2: Add a controller that writes to the Datastore

> Copy a screenshot of Datastore Studio with the written entity into the report.



Genre: book

Résultats de la requête

Nom/ID	author	title
id=5634161670881280	John Steinbeck	The grapes of wrath

ID de la base de données	(default)
Clé	book id:5634161670881280
Clé littérale	Key(book, 5634161670881280)
Clé sécurisée pour les URL	ahJlfmNsZGxhYmdhZS00MjEyMTNyEQsSBGJvb2sYgICA6NeHgQoM

author
John Steinbeck

title
The grapes of wrath

Task 3: Develop a controller to write arbitrary entities into the Datastore

> Copy a code listing of your app into the report.

```
//... Code from earlier

    @GetMapping("/dswrite")
    public String writeEntityToDatastore(@RequestParam Map<String, String>
queryParameters) {
        String kind = queryParameters.get("_kind");
        StringBuilder message = new StringBuilder();
        if(kind == null){
            message.append("The kind attribute is mandatory!");
            return message.toString();
        }

        //if we got a kind
        KeyFactory keyFactory = datastore.newKeyFactory().setKind(kind);
        Key key = null;
        if(queryParameters.containsKey("_key")){
            key = keyFactory.newKey(queryParameters.get("_key"));
        } else {
            key = datastore.allocateId(keyFactory.newKey());
        }

        //Create the builder
        Entity.Builder builder = Entity.newBuilder(key);

        for(var param : queryParameters.entrySet()){
            if(!param.getKey().equals("_key") &&
!param.getKey().equals("_kind")){
                builder.set(param.getKey(), param.getValue());
            }
        }

        message.append("Writing entity to Datastore\n");
        datastore.put(builder.build());

        return message.toString();
    }
}
```

Task 4: Test the performance of datastore writes

> For each performance test include a graph of the load testing tool and copy three screenshots of the App Engine instances view (graph of requests by type, graph of number of instances, graph of latency) into the report.

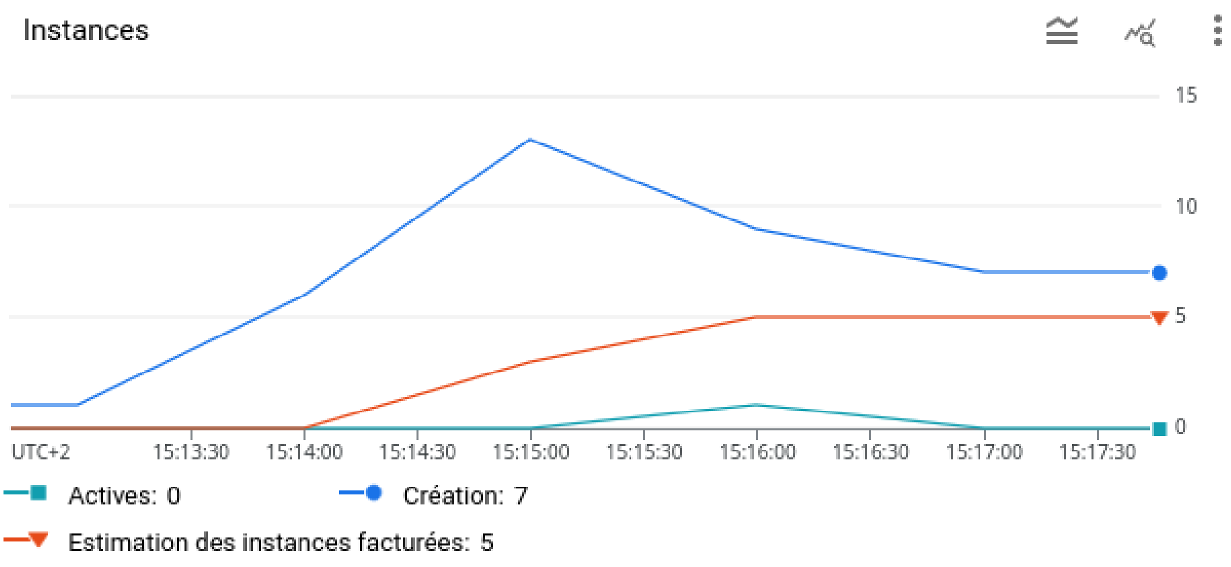
Hello world handler:

Vegeta command:

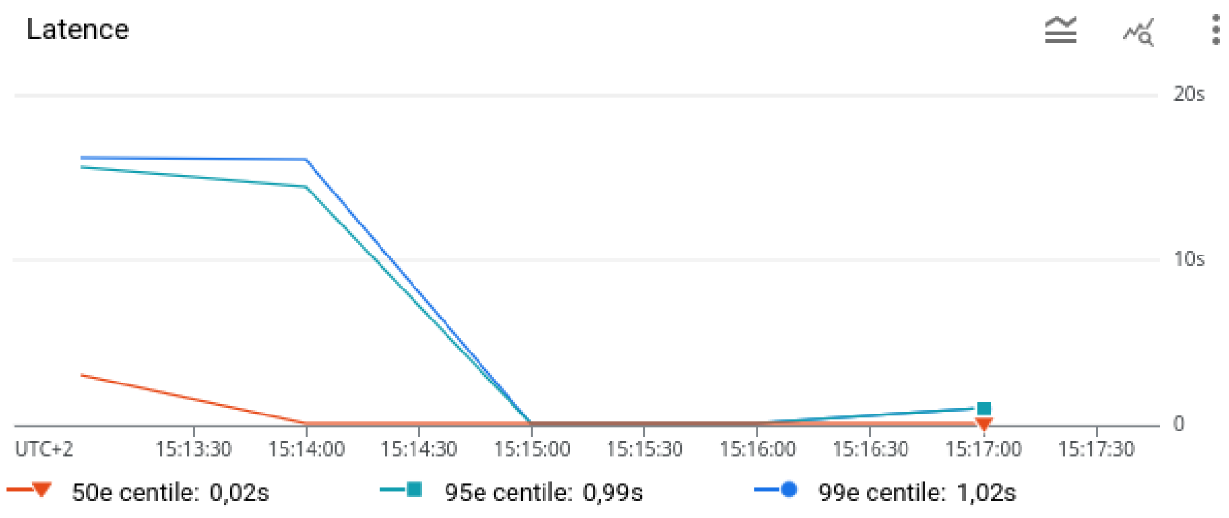
```
echo "GET https://cldlabgae-421213.ew.r.appspot.com/" | vegeta attack -
duration=120s | tee helloWorldResults.bin | vegeta report
```

On AppEngine:

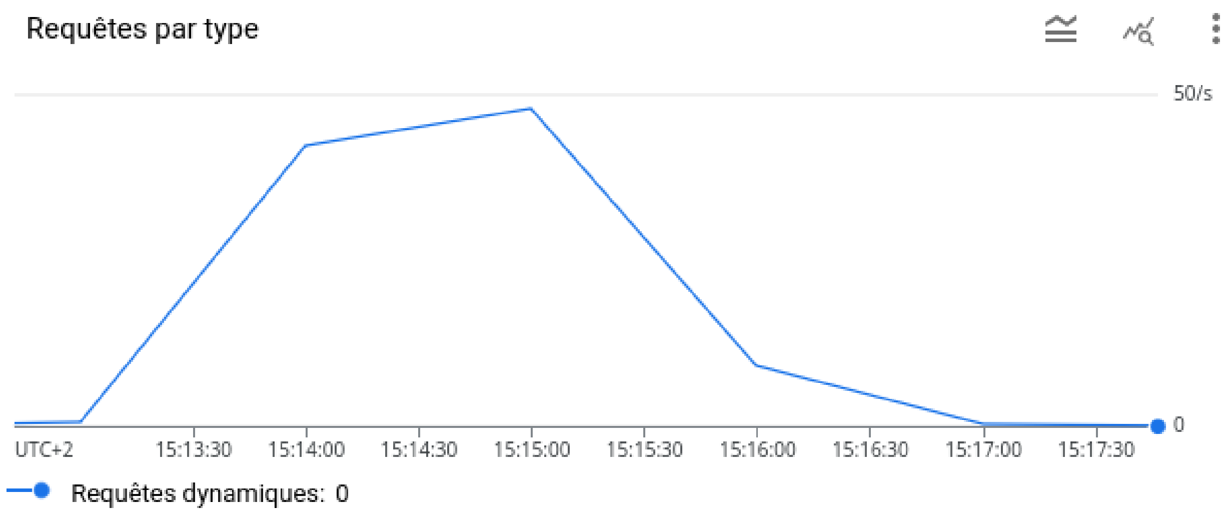
- Instances view



- Latency view

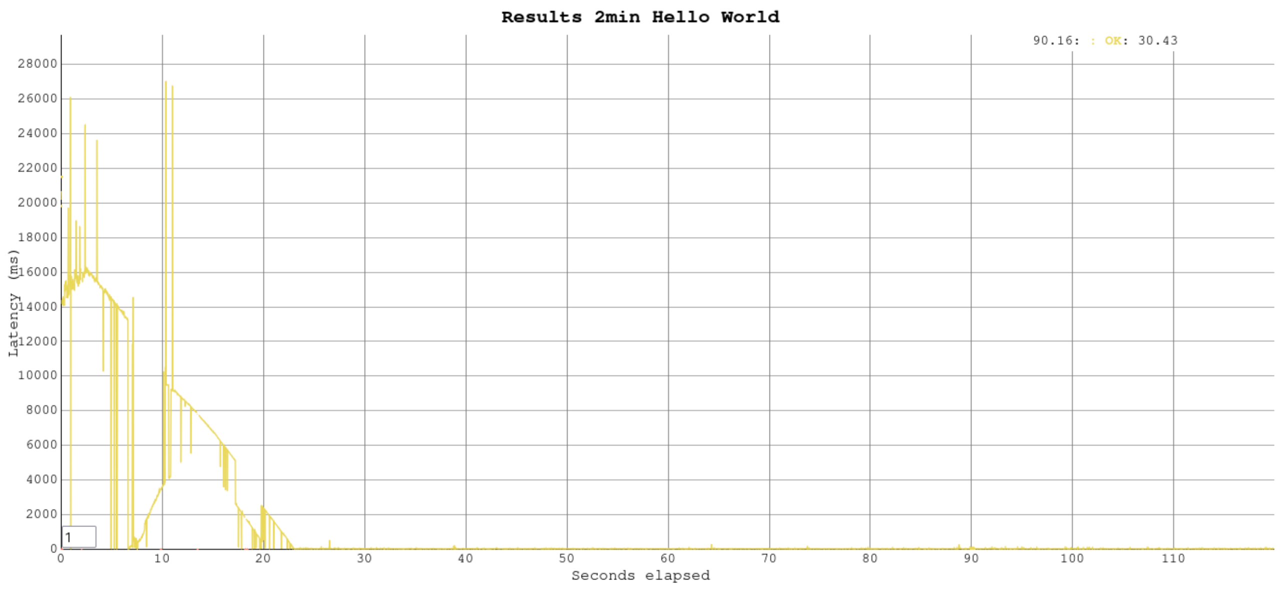


- Requests by type view



On testing tool vegeta:

- Graph



- Console

```
Requests      [total, rate, throughput]         6000, 50.01, 49.85
Duration      [total, attack, wait]             2m0s, 2m0s, 26.923ms
Latencies     [min, mean, 50, 90, 95, 99, max]  181.962µs, 1.386s,
30.488ms, 5.901s, 13.634s, 15.844s, 27.032s
Bytes In      [total, mean]                     77766, 12.96
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           99.70%
Status Codes  [code:count]                      0:18  200:5982
Error Set:
Get "https://cldlabgae-421213.ew.r.appspot.com/": dial tcp 0.0.0.0:0->
[2a00:1450:400a:803::2014]:443: connect: network is unreachable
```

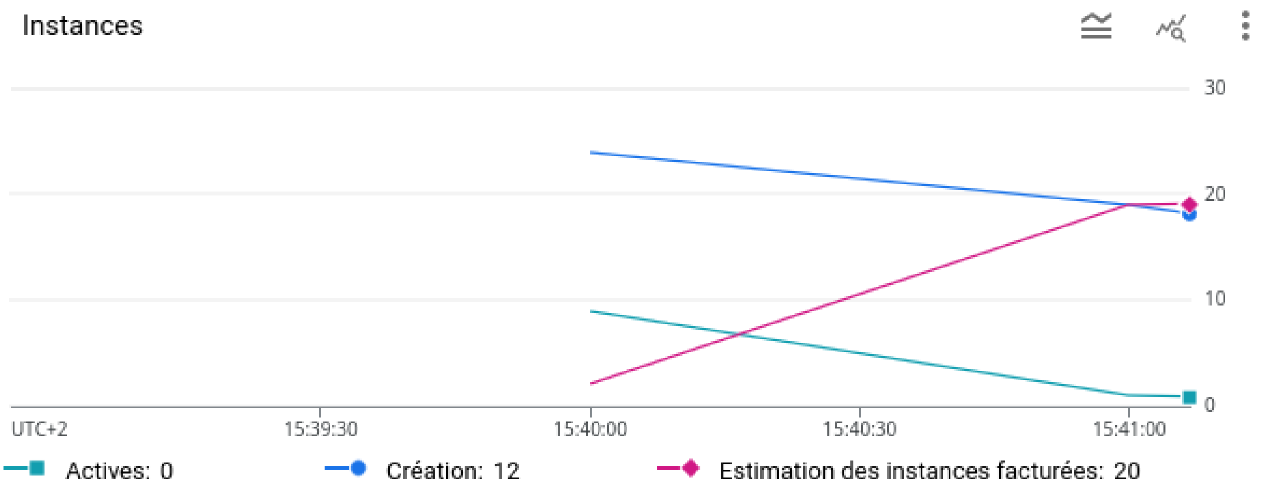
Datastore write handler:

Vegeta command:

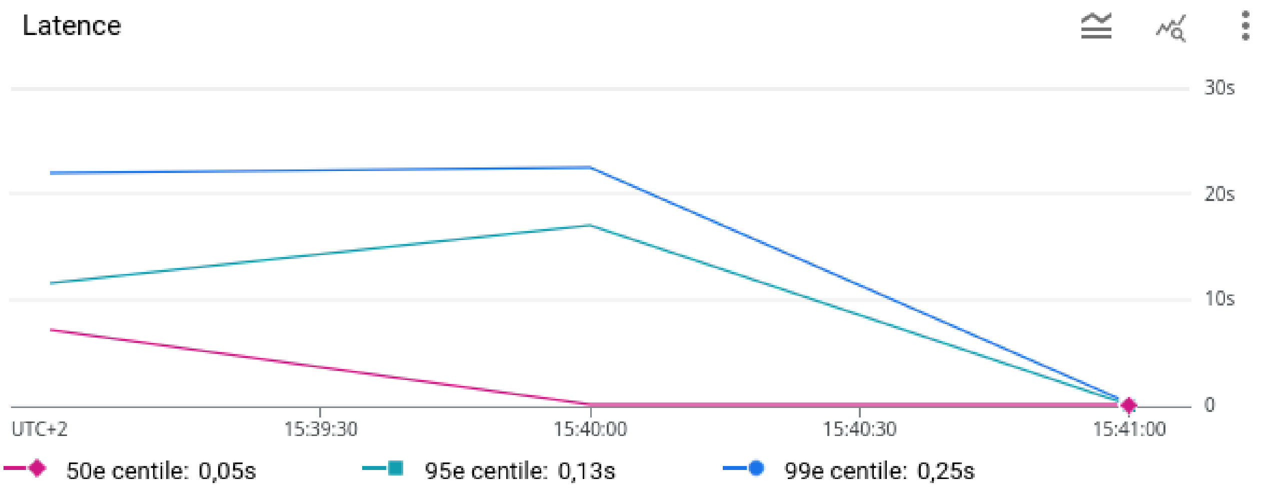
```
echo "GET https://cldlabgae-421213.ew.r.appspot.com/dswrite?
_kind=book&author=John%20Steinbeck&title=The%20Grapes%20of%20Wrath" |
vegeta attack -duration=120s | tee dswriteResults.bin | vegeta report
```

On AppEngine:

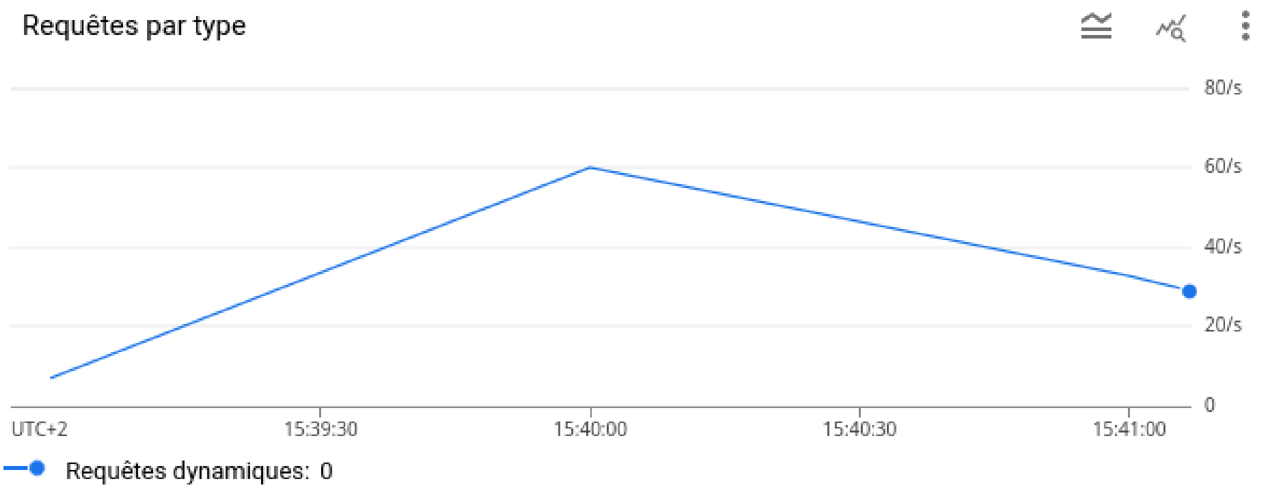
- Instances view



- Latency view

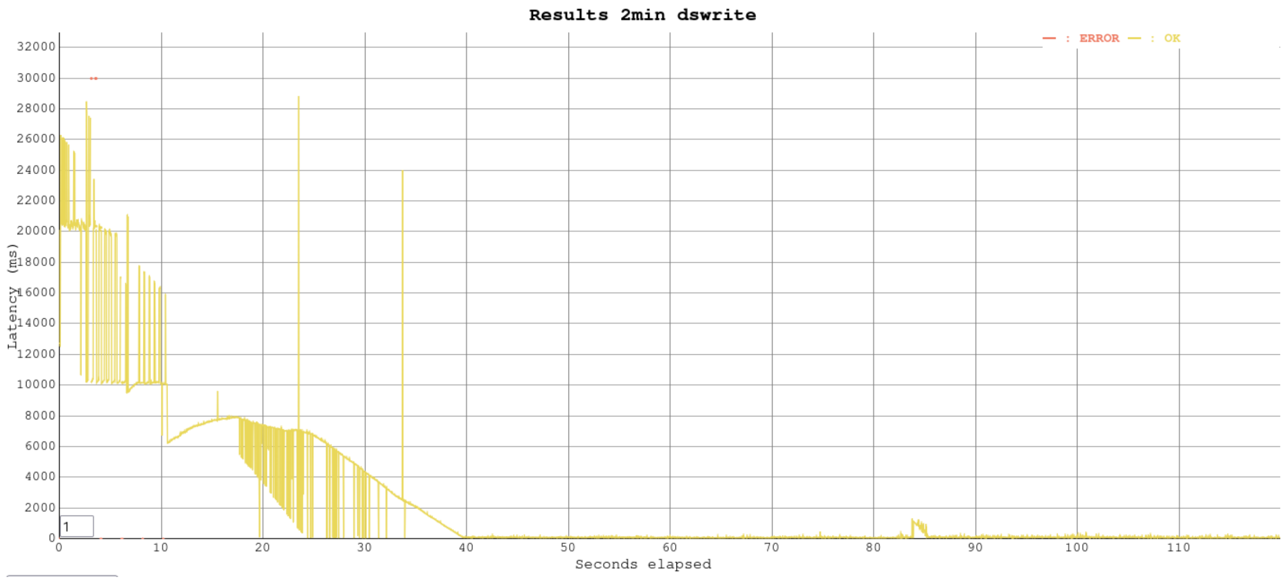


- Requests by type view



On testing tool vegeta:

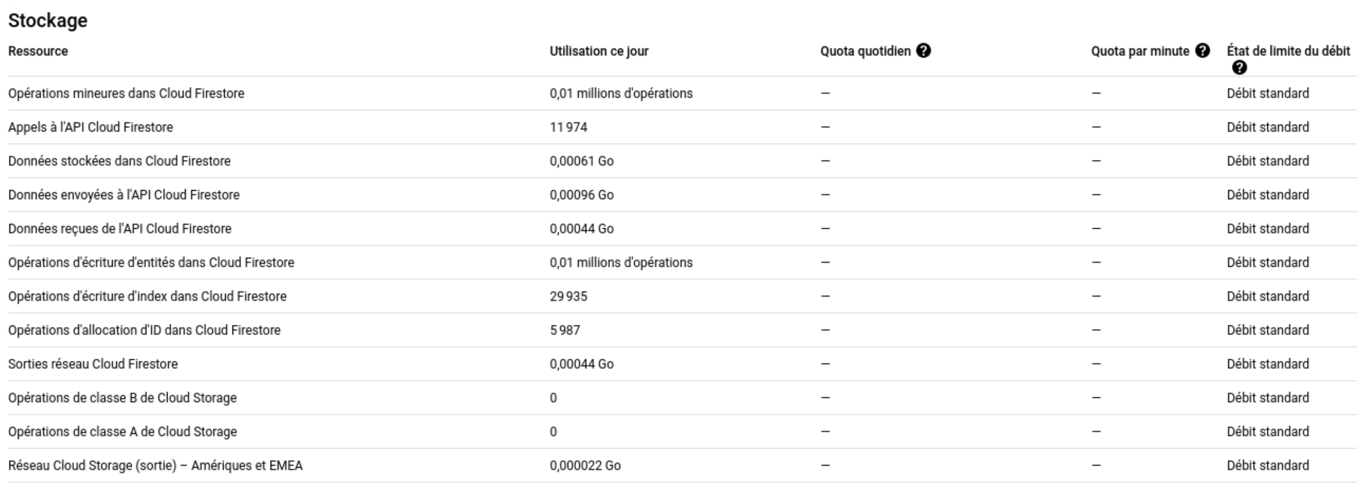
- Graph



- Console

```
Requests      [total, rate, throughput]         6000, 50.01, 49.85
Duration      [total, attack, wait]             2m0s, 2m0s, 68.945ms
Latencies     [min, mean, 50, 90, 95, 99, max]  209.4µs, 2.569s, 88.844ms,
7.82s, 10.224s, 20.561s, 30.001s
Bytes In      [total, mean]                     167552, 27.93
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           99.73%
Status Codes  [code:count]                      0:16  200:5984
Error Set:
Get "https://cldlabgae-421213.ew.r.appspot.com/dswrite?
_kind=book&author=John%20Steinbeck&title=The%20Grapes%20of%20Wrath": dial
tcp 0.0.0.0:0->[2a00:1450:400a:803::2014]:443: connect: network is
unreachable
Get "https://cldlabgae-421213.ew.r.appspot.com/dswrite?
_kind=book&author=John%20Steinbeck&title=The%20Grapes%20of%20Wrath":
net/http: request canceled (Client.Timeout exceeded while awaiting headers)
```

Quotas at the end:



Stockage

Ressource	Utilisation ce jour	Quota quotidien	Quota par minute	État de limite du débit
Opérations mineures dans Cloud Firestore	0,01 millions d'opérations	–	–	Débit standard
Appels à l'API Cloud Firestore	11 974	–	–	Débit standard
Données stockées dans Cloud Firestore	0,00061 Go	–	–	Débit standard
Données envoyées à l'API Cloud Firestore	0,00096 Go	–	–	Débit standard
Données reçues de l'API Cloud Firestore	0,00044 Go	–	–	Débit standard
Opérations d'écriture d'entités dans Cloud Firestore	0,01 millions d'opérations	–	–	Débit standard
Opérations d'écriture d'index dans Cloud Firestore	29 935	–	–	Débit standard
Opérations d'allocation d'ID dans Cloud Firestore	5 987	–	–	Débit standard
Sorties réseau Cloud Firestore	0,00044 Go	–	–	Débit standard
Opérations de classe B de Cloud Storage	0	–	–	Débit standard
Opérations de classe A de Cloud Storage	0	–	–	Débit standard
Réseau Cloud Storage (sortie) – Amériques et EMEA	0,000022 Go	–	–	Débit standard

> What average response times do you observe in the test tool for each controller?

For the Hello world handler, we have a mean latency of 1.386s on vegeta and 2.569s for the dswrite handler.

> Compare the response times shown by the test tool and the App Engine console. Explain the difference.

The difference is that we made our test on the span of 2 minutes where the measures on the google console is made every minute. But even with those, the values were quite close !

> Let’s suppose you become suspicious that the algorithm for the automatic scaling of instances is not working correctly. Imagine a way in which the algorithm could be broken. Which measures shown in the console would you use to detect this failure?

The monitoring or scaling action of the autoscaling algorithm could be broken if:

1. The latency is going through the roof. We can suspect a malfunction in the algorithm because it means that it doesn't respond correctly to an upsurge of requests by adding more instances.

2. In addition to the high latency, we can check if new instances are not being created while the load is high. If not, there's an issue with the scaling algorithm!!

3. The number of requests by type view would not help us in any way because this metric cannot highlight any problem with our algorithm. We would see a drop in the number of requests handled, but nothing that could allow us to confirm if it's a problem coming from our scaling algorithm specifically.

4. For the traffic metrics, while we cannot use them alone, by analyzing them together with the "number of instances" metric, we could detect a scaling issue. If traffic is increasing but the number of instances remains static, that would indicate the scaling algorithm is not allocating new instances despite the rise of demands.

Those would be the easiest ways to detect if the algorithm isn't working the way it should.