Software at Scale

CSSE6400

Brae Webb

March 28, 2022

How many concurrent users can your software handle?

How many concurrent users can your software handle?

Maybe *400*? Maximum.

How many concurrent users can your software handle?

Maybe 400[1]? Maximum.

---

[1]HTTP server on a t2.micro EC2 instance

Definition 1. Stress Testing

Measure the robustness of software by pushing usage to an extreme.

Demonstration

Let's build 'hello world'

Our Goal

Hello world from *your name here*

My Goal

```
» cat hello-server.tf
```

```
1  resource "aws_instance" "hello-server" {
2      ami = "ami-04902260ca3d33422"
3      instance_type = "t2.micro"
4  }
```

```
» cat hello-server.tf

1   resource "aws_instance" "hello-server" {
2       ami = "ami-04902260ca3d33422"
3       instance_type = "t2.micro"

5       user_data = file("./setup.sh")
6   }
```

```
» cat setup.sh
```

```bash
#!/bin/bash
yum -y install httpd
systemctl enable httpd
systemctl start httpd
echo '<html><title>Hello, world!</title><h1>Hello world from Brae</h1></html>' > /
    var/www/html/index.html
```

```
» cat hello-server.tf
```

```
1   resource "aws_instance" "hello-server" {
2       ami = "ami-04902260ca3d33422"
3       instance_type = "t2.micro"

5       user_data = file("./setup.sh")

7       security_groups = [
8           aws_security_group.hello-server.name
9       ]
10  }
```

```
» cat hello-server.tf

1   resource "aws_instance" "hello-server" {
2       ami = "ami-04902260ca3d33422"
3       instance_type = "t2.micro"

5       user_data = file("./setup.sh")

7       security_groups = [
8           aws_security_group.hello-server.name
9       ]

11      tags = {
12          Name = "hello-server"
13      }
14  }
```

Starting the server

```
1  >> terraform init
2  >> terraform plan
3  >> terraform apply
```

Before

Hello world from Brae

How much traffic can this website handle?

```
» cat stress-test.js
```

```javascript
1   import http from 'k6/http';
2   import { check, sleep } from 'k6';

4   const IP = "http://3.6.9.12/";
5   export default function() {
6       const res = http.get(IP);
7       check(res, { 'status was 200': (r) => r.status == 200 });
8       sleep(1);
9   }
```

```
» cat stress-test.js
```

```
1   import http from 'k6/http';
2   import { check, sleep } from 'k6';

4   const IP = "http://3.6.9.12/";
5   export const options = {
6       stages: [
7           { duration: '2m', target: 100 },
8       ],
9   };
10  export default function() {
11      const res = http.get(IP);
12      check(res, { 'status was 200': (r) => r.status == 200 });
13      sleep(1);
14  }
```

Run the tests

```
>> k6 run stress-test.js
```

Looks good so far

```
1   status was 200
2     100% - 347867 / 0

4   checks.........................: 100%
5   data_received..................: 100 MB 44 kB/s
6   data_sent......................: 27 MB 12 kB/s
7   iterations.....................: 347997 152.552084/s
8   vus............................: 1 min=1 max=400
```

Let's upgrade the traffic

```
» cat stress-test.js
```

```
1  export const options = {
2     stages: [
3        { duration: '2m', target: 100 },
4        { duration: '5m', target: 100 },
5        { duration: '2m', target: 200 },
6        { duration: '5m', target: 200 },
7        { duration: '2m', target: 300 }, // around the breaking point
8        { duration: '5m', target: 300 },
9        { duration: '2m', target: 400 }, // beyond the breaking point
10       { duration: '5m', target: 400 },
11       { duration: '2m', target: 0 }, // scale down
12    ],
13 };
```
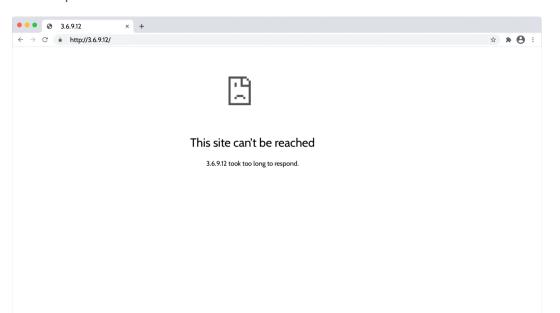
And run the tests again

```
1  >> k6 run stress-test.js
```

Oh no…

```
1  status was 200
2  99% - 347867 / 130

4  checks...........................: 99.96%
5  data_received....................: 100 MB 44 kB/s
6  data_sent........................: 27 MB 12 kB/s
7  iterations.......................: 347997 152.552084/s
8  vus..............................: 1 min=1 max=400
```

Back to square one

Question

How can we fix this?

Question

How can we fix this?

Answer

More servers?

```
» cat hello-server.tf
```

```
1   resource "aws_instance" "hello-server" {
2       ami = "ami-04902260ca3d33422"
3       instance_type = "t2.micro"

5       user_data = file("./setup.sh")

7       security_groups = [
8           aws_security_group.hello-server.name
9       ]

11      tags = {
12          Name = "hello-server"
13      }
14  }
```

```
» cat hello-scale.tf
```

```
1  resource "aws_instance" "hello-server" {
2    count = 4

4    ami = "ami-04902260ca3d33422"
5    instance_type = "t2.micro"
6    user_data = file("${path.module}/setup.sh")

8    security_groups = [
9      aws_security_group.hello-server.name
10   ]

12   tags = {
13     Name = "hello-server-${count.index}"
14   }
15 }
```

Definition 2. Target Group

A collection of EC2 instances.
More specifically, a collection of network connection points to EC2 instances.

An empty HTTP target group

```
» cat hello-scale.tf
```

```
1  resource "aws_lb_target_group" "hello-target" {
2    name = "hello-target-group"
3    port = 80
4    protocol = "HTTP"
5    vpc_id = aws_security_group.hello-server.vpc_id
6  }
```

Definition 3. Health Check

Monitors attributes of hardware or software to detect deficiencies.

Add a health check

```
» cat hello-scale.tf
```

```
1  resource "aws_lb_target_group" "hello-target" {
2    name = "hello-target-group"
3    port = 80
4    protocol = "HTTP"
5    vpc_id = aws_security_group.hello-server.vpc_id
6
7    health_check {
8      port = 80
9      protocol = "HTTP"
10     timeout = 5
11     interval = 10
12   }
13 }
```

Add our instances to the target group

```
» cat hello-scale.tf
```

```
1  resource "aws_lb_target_group_attachment" "hello-target-link" {
2    count = length(aws_instance.hello-server)
3    target_group_arn = aws_lb_target_group.hello-target.arn
4    target_id = aws_instance.hello-server[count.index].id
5    port = 80
6  }
```

> **Definition 4. Load Balancer**
>
> A networking tool to route and distribute traffic to targets.

Create a load balancer

```
» cat hello-scale.tf
1  data "aws_subnet_ids" "nets" {
2      vpc_id = aws_security_group.hello-server.vpc_id
3  }
4
5  resource "aws_lb" "hello-balancer" {
6    name = "hello-balancer"
7    internal = false
8    load_balancer_type = "application"
9    subnets = aws_subnet_ids.nets.ids
10   security_groups = [
11     aws_security_group.hello-server.name
12   ]
13 }
```

Route load balancer traffic to the target group

```
» cat hello-scale.tf
```

```
1  resource "aws_lb_listener" "app" {
2    load_balancer_arn = aws_lb.hello-balancer.arn
3    port = "80"
4    protocol = "HTTP"

6    default_action {
7      type = "forward"
8      target_group_arn = aws_lb_target_group.hello-target.arn
9    }
10 }
```

We're live!

Use *k6* to determine the new *load limits*