Software at Scale

CSSE6400

Brae Webb

March 28, 2022

How many concurrent users can your software handle?

Question

How many concurrent users can your software handle?

Answer

Maybe *400*? Maximum.

How many concurrent users can your software handle?

Maybe *400*[1]? Maximum.

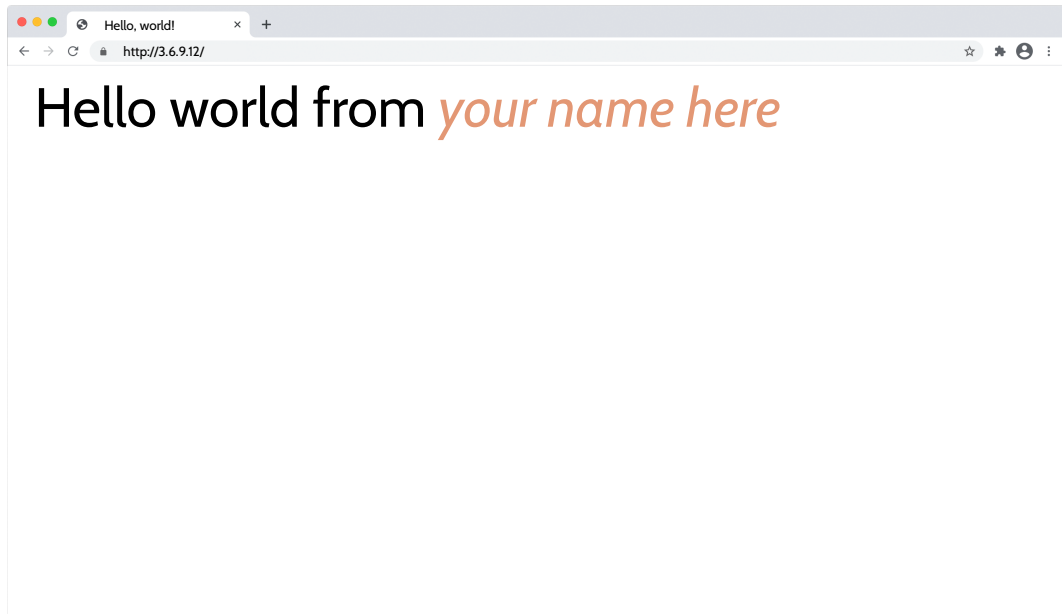
[1]HTTP server on a t2.micro EC2 instance

> **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*

Hello world from Brae

```
» cat hello-server.tf

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

```
» cat hello-server.tf

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

    user_data = file("${path.module}/setup.sh")
}
```

```
» cat setup.sh

#!/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("${path.module}/setup.sh")

7      associate_public_ip_address = true
8      subnet_id = module.network.subnets[0].public.id
9      vpc_security_group_ids = [
10         module.network.http-port.id
11     ]
12 }
```

```
» cat hello-server.tf

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

    user_data = file("${path.module}/setup.sh")

    associate_public_ip_address = true
    subnet_id = module.network.subnets[0].public.id
    vpc_security_group_ids = [
        module.network.http-port.id
    ]

    tags = {
        Name = "hello-server"
    }
}
```

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

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

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

```
» cat stress-test.js

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

const IP = "http://3.6.9.12/";
export const options = {
    stages: [
        { duration: '2m', target: 100 },
    ],
};
export default function() {
    const res = http.get(IP);
    check(res, { 'status was 200': (r) => r.status == 200 });
    sleep(1);
}
```

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
```

```js
export const options = {
    stages: [
        { duration: '2m', target: 100 },
        { duration: '5m', target: 100 },
        { duration: '2m', target: 200 },
        { duration: '5m', target: 200 },
        { duration: '2m', target: 300 }, // around the breaking point
        { duration: '5m', target: 300 },
        { duration: '2m', target: 400 }, // beyond the breaking point
        { duration: '5m', target: 400 },
        { duration: '2m', target: 0 }, // scale down
    ],
};
```

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?

How can we fix this?

More servers?

```
» cat hello-server.tf

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

    user_data = file("${path.module}/setup.sh")

    associate_public_ip_address = true
    subnet_id = module.network.subnets[0].public.id
    vpc_security_group_ids = [
        module.network.http-port.id
    ]

    tags = {
        Name = "hello-server"
    }
}
```

```
» cat hello-scale.tf
```

```terraform
resource "aws_instance" "hello-server" {
  count = 4

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

  associate_public_ip_address = true
  subnet_id = module.network.subnets[count.index].public.id
  vpc_security_group_ids = [
    module.network.http-port.id
  ]

  tags = {
    Name = "hello-server-${count.index}"
  }
}
```

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 = module.network.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 = module.network.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  resource "aws_lb" "hello-balancer" {
2    name = "hello-balancer"
3    internal = false
4    load_balancer_type = "application"
5    subnets = [
6      module.network.subnets[0].public.id,
7      module.network.subnets[1].public.id,
8      module.network.subnets[2].public.id,
9      module.network.subnets[3].public.id
10   ]
11   security_groups = [
12     module.network.http-port.id
13   ]
14 }
```

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!
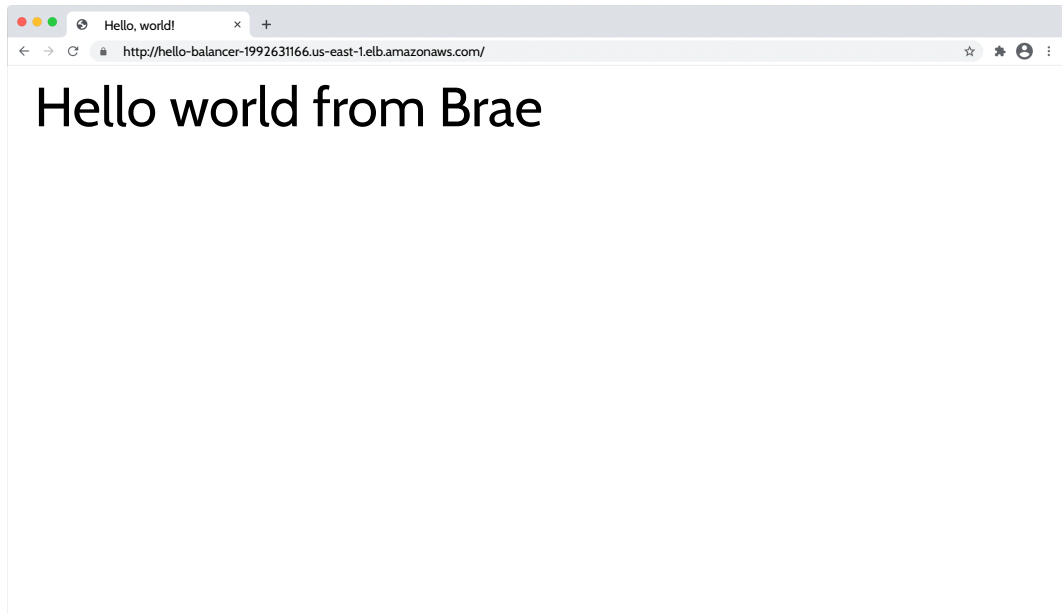
Exercise

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