Software at Scale Software Architecture

Brae Webb

March 20, 2023

How many concurrent users can your software handle?

How many concurrent users can your software handle?

Answer

Maybe 400? Maximum.

How many concurrent users can your software handle?

Answer

Maybe 400^{1} ? Maximum.

¹HTTP 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



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("./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

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

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

   security_groups = [
      aws_security_group.hello-server.name
]
```

```
resource "aws_instance" "hello-server" {
   ami = "ami-04902260ca3d33422"
   instance_type = "t2.micro"
   user_data = file("./setup.sh")
   security_groups = [
       aws_security_group.hello-server.name
   tags = {
       Name = "hello-server"
```

>> cat hello-server.tf

5

11

12

Starting the server

- >> terraform init
- >> terraform plan
- >> terraform apply

Before





This site can't be reached

3.6.9.12 took too long to respond.

After



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() {
10
        const res = http.get(IP);
11
        check(res, { 'status was 200': (r) => r.status == 200 });
12
        sleep(1);
13
14
```

Run the tests

>> k6 run stress-test.js

Looks good so far

Let's upgrade the traffic

10

11

12

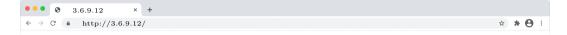
```
>> cat stress-test.js
export const options = {
   stages: [
       { duration: '2m', target: 100 },
       { duration: '5m', target: 100 },
       { duration: '2m', target: 200 },
       { duration: '5m', target: 200 },
       { duration: '2m', target: 300 }, // around the breaking point
       { 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

>> k6 run stress-test.js

Oh no...

Back to square one





This site can't be reached

3.6.9.12 took too long to respond.

How can we fix this?

How can we fix this?

Answer

More servers?

```
resource "aws_instance" "hello-server" {
   ami = "ami-04902260ca3d33422"
   instance_type = "t2.micro"
   user_data = file("./setup.sh")
   security_groups = [
       aws_security_group.hello-server.name
   tags = {
       Name = "hello-server"
```

>> cat hello-server.tf

5

11

12

```
>> cat hello-scale.tf
resource "aws_instance" "hello-server" {
 count = 4
 ami = "ami - 04902260ca3d33422"
 instance_type = "t2.micro"
 user_data = file("${path.module}/setup.sh")
 security_groups = [
   aws_security_group.hello-server.name
 tags = {
   Name = "hello-server-${count.index}"
```

8

10

12

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

resource "aws_lb_target_group" "hello-target" {
  name = "hello-target-group"
  port = 80
  protocol = "HTTP"
  vpc_id = aws_security_group.hello-server.vpc_id
```

Definition 3. Health Check

Monitors attributes of hardware or software to detect deficiencies.

Add a health check

10

```
>> cat hello-scale.tf
resource "aws_lb_target_group" "hello-target" {
 name = "hello-target-group"
 port = 80
 protocol = "HTTP"
 vpc_id = aws_security_group.hello-server.vpc_id
 health_check {
   port = 80
   protocol = "HTTP"
   timeout = 5
   interval = 10
```

Add our instances to the target group

```
>>> cat hello-scale.tf

resource "aws_lb_target_group_attachment" "hello-target-link" {
  count = length(aws_instance.hello-server)
  target_group_arn = aws_lb_target_group.hello-target.arn
  target_id = aws_instance.hello-server[count.index].id
  port = 80
```

Definition 4. Load Balancer

A networking tool to route and distribute traffic to targets.

Create a load balancer

10

```
>> cat hello-scale.tf
data "aws_subnet_ids" "nets" {
   vpc_id = aws_securitv_group.hello-server.vpc_id
resource "aws_lb" "hello-balancer" {
 name = "hello-balancer"
 internal = false
 load_balancer_type = "application"
 subnets = aws_subnet_ids.nets.ids
 security_groups = [
   aws_security_group.hello-server.name
```

Route load balancer traffic to the target group

default_action {
 type = "forward"

10

```
>> cat hello-scale.tf

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

target_group_arn = aws_lb_target_group.hello-target.arn

We're live!



Hello world from Brae

Exercise

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