

# Software at Scale

CSSE6400

Brae Webb

March 21, 2022

Question

How many concurrent users can your software handle?

### Question

How many concurrent users can your software handle?

### Answer

Maybe **400**? Maximum.

### Question

How many concurrent users can your software handle?

### Answer

Maybe 400<sup>1</sup>? Maximum.

---

<sup>1</sup>HTTP server on a t2.micro EC2 instance

### Definition 1. Stress Testing

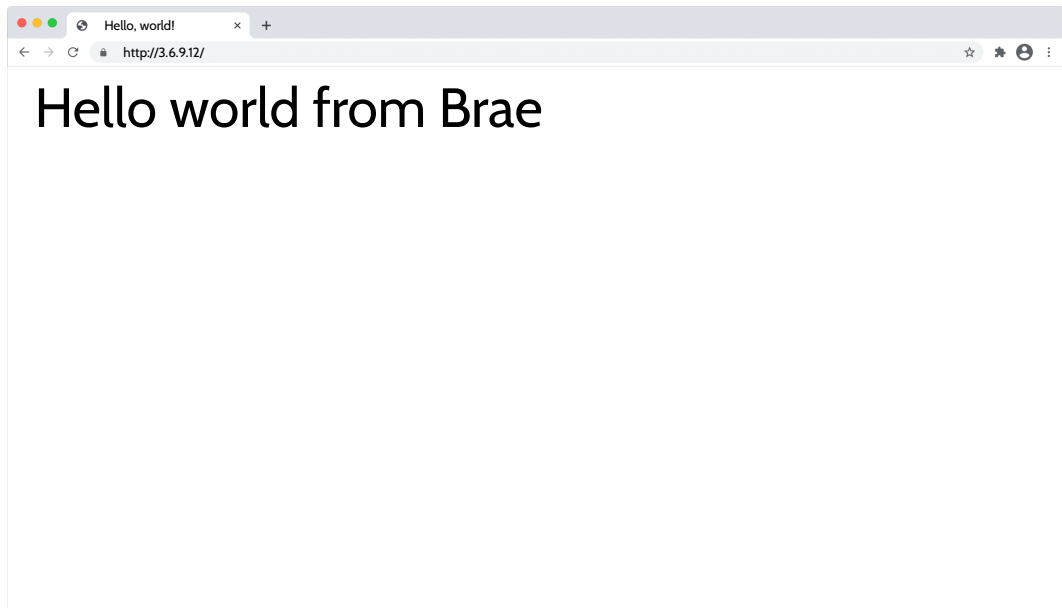
Measure the robustness of software by pushing usage to an extreme.

Demonstration

Let's build 'hello world'

## Our Goal







```
» 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("${path.module}/setup.sh")  
6 }
```

```
» cat setup.sh
```

```
1  #!/bin/bash
2  yum -y install httpd
3  systemctl enable httpd
4  systemctl start httpd
5  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
```

```
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     ]

13     tags = {
14         Name = "hello-server"
15     }
16 }
```

## Starting the server

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

# Before



After





Question

How much traffic can this website handle?

```
» 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 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

```
1 >> 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("${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     ]

13     tags = {
14         Name = "hello-server"
15     }
16 }
```

```
» 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     associate_public_ip_address = true  
9     subnet_id = module.network.subnets[count.index].public.id  
10    vpc_security_group_ids = [  
11        module.network.http-port.id  
12    ]  
  
14    tags = {  
15        Name = "hello-server-${count.index}"  
16    }  
17 }
```

## 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"
5
6     default_action {
7         type = "forward"
8         target_group_arn = aws_lb_target_group.hello-target.arn
9     }
10 }
```

We're live!



# Hello world from Brae

Exercise

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

## References

- [1] Martin Fowler.  
Software architecture guide.  
<https://martinfowler.com/architecture/>, August 2019.
- [2] Matthias Galster and Samuil Angelov.  
What makes teaching software architecture difficult?  
In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 356–359. Association for Computing Machinery, 2016.
- [3] Robert C. Martin.  
Design principles and design patterns.  
[https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles\\_and\\_Patterns.pdf](https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf), 2000.  
Accessed: 2022-01-10.



- [4] Philippe Kruchten, Rafael Capilla, and Juan Carlos Duenas.  
The decision view's role in software architecture practice.  
*IEEE software*, 26(2):36–42, 2009.
- [5] Michael Nygard.  
Documenting architecture decisions.  
[https://cognitect.com/blog/2011/11/15/  
documenting-architecture-decisions](https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions), November 2011.  
Accessed: 2022-01-27.
- [6] Olaf Zimmermann.  
Architectural decisions – the making of.  
[https:  
//ozimmer.ch/practices/2020/04/27/ArchitectureDecisionMaking.html](https://ozimmer.ch/practices/2020/04/27/ArchitectureDecisionMaking.html),  
March 2021.  
Accessed: 2022-02-02.

[7] Eli Perkins.

Why write adrs.

<https://github.blog/2020-08-13-why-write-adrs/>, August 2020.

Accessed: 2022-02-02.

[8] Richard Thomas.

Architectural decision records.

February 2022.

<https://csse6400.uqcloud.net/handouts/adr.pdf>.

[9] Eric Boersma.

7 application security principles you need to know.

[https://www.cprime.com/resources/blog/](https://www.cprime.com/resources/blog/security-by-design-7-principles-you-need-to-know/)

[security-by-design-7-principles-you-need-to-know/](https://www.cprime.com/resources/blog/security-by-design-7-principles-you-need-to-know/), October 2020.

[10] Jerome H Saltzer and Michael D Schroeder.

The protection of information in computer systems.

*Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

- [11] Morrie Gasser.  
*Building a Secure Computer System*, pages 35–44.  
Van Nostrand Reinhold Company, January 1988.
- [12] John Viega and Gary R McGraw.  
*Building Secure Software: How to Avoid Security Problems the Right Way*, pages 91–113.  
Addison-Wesley Professional, September 2001.
- [13] Michael Howard and David LeBlanc.  
*Security Principles To Live By*, page 64.  
Microsoft Press Redmond, Wash., December 2002.
- [14] Michael Gegick and Sean Barnum.  
*Failing securely*.  
<https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/failing-securely>, December 2005.

[15] Santosh Janardhan.

More details about the October 4 outage.

<https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/>, October 2021.

[16] C. E. Shannon.

Communication theory of secrecy systems.

*The Bell System Technical Journal*, 28(4):656–715, 1949.

[17] Sam Manjarres.

2021 world password day: How many will be stolen this year?

<https://www.secplicity.org/2021/05/04/2021-world-password-day-how-many-will-be-stolen-this-year/>, May 2021.

[18] Jerome H. Saltzer.

Protection and the control of information sharing in multics.

*Communications of the ACM*, 17(7):388–402, July 1974.

- [19] Emma Roth.  
Open source developer corrupts widely-used libraries, affecting tons of projects.  
[https://www.theverge.com/2022/1/9/22874949/](https://www.theverge.com/2022/1/9/22874949/developer-corrupts-open-source-libraries-projects-affected)  
developer-corrupts-open-source-libraries-projects-affected, January 2022.
- [20] Brian Foote and Joseph Yoder.  
Big ball of mud.  
*Pattern languages of program design*, 4:654–692, 1997.
- [21] Gulsah.  
How to avoid spaghetti code.  
<https://tech.zensurance.com/posts/spaghetti-code>, November 2020.  
note = "Accessed: 2022-02-18".
- [22] Jeffrey Dean and Sanjay Ghemawat.  
Mapreduce: Simplified data processing on large clusters.  
In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*,  
pages 137–150, San Francisco, CA, 2004.

[23] David J. DeWitt and Michael Stonebraker.

Mapreduce: A major step backwards.

<https://dsf.berkeley.edu/cs286/papers/backwards-vertica2008.pdf>,  
January 2008.

[24] henszey.

Smallest x86 ELF hello world.

<http://timelessname.com/elfbin/>.

[25] Changhui Xu.

Docker: From scratch.

<https://codeburst.io/docker-from-scratch-2a84552470c8>, July 2020.

[26] Philippe Kruchten.

Architectural blueprints — the ‘4+1’ view model of software architecture.

*IEEE software*, 12(6):42–50, 1995.

[https:](https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf)

[//www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf](https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf).

[27] Richard Thomas and Brae Webb.

*Architectural views.*

February 2022.

<https://csse6400.uqcloud.net/handouts/views.pdf>.

[28] Alexander Shvets.

*Observer.*

<https://refactoring.guru/design-patterns/observer>.

note = "Accessed: 2022-02-18".

[29] Tom Hilburn, Alice Squires, Heidi Davidz, and Richard Turner.

*Federal aviation administration (faa) advanced automation system (aas).*

[https://www.sebokwiki.org/wiki/Federal\\_Aviation\\_Administration\\_\(FAA\)\\_Advanced\\_Automation\\_System\\_\(AAS\)](https://www.sebokwiki.org/wiki/Federal_Aviation_Administration_(FAA)_Advanced_Automation_System_(AAS)), October 2021.

Example from the *Guide to the Systems Engineering Body of Knowledge*

[https://www.sebokwiki.org/w/index.php?title=Guide\\_to\\_the\\_Systems\\_Engineering\\_Body\\_of\\_Knowledge\\_\(SEBoK\)&oldid=63222](https://www.sebokwiki.org/w/index.php?title=Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)&oldid=63222).

- [30] Raymond Lister, Colin Fidge, and Donna Teague.  
*Further evidence of a relationship between explaining, tracing and writing skills in introductory programming.*  
In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, page 161–165, New York, NY, USA, 2009. Association for Computing Machinery.
- [31] Raymond Lister.  
*Concrete and other neo-piagetian forms of reasoning in the novice programmer.*  
In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*, ACE '11, page 9–18, AUS, 2011. Australian Computer Society, Inc.
- [32] Len Bass, Paul Clements, and Rick Kazman.  
*Software Architecture in Practice.*  
Addison-Wesley Professional, 3rd edition, September 2012.



- [33] Len Bass, Paul Clements, and Rick Kazman.  
*Software Architecture in Practice.*  
Addison-Wesley, 4th edition, August 2021.
- [34] David Garlan, Felix Bachmann, James Ivers, Judith Stafford, Len Bass, Paul Clements, and Paulo Merson.  
*Documenting Software Architectures: Views and Beyond.*  
Addison-Wesley Professional, 2nd edition, 2010.
- [35] Mark Richards and Neal Ford.  
*Fundamentals of Software Architecture: An Engineering Approach.*  
O'Reilly Media, Inc., January 2020.
- [36] Sam Newman.  
*Building Microservices.*  
O'Reilly Media, Inc., February 2015.

- [37] Simon Brown.  
*Software Architecture for Developers - Volume 2.*  
Leanpub, January 2022.  
<https://leanpub.com/visualising-software-architecture>.
- [38] Philippe Kruchten.  
*The Rational Unified Process: An Introduction.*  
Addison-Wesley Professional, 2004.
- [39] Gerald M. Weinberg.  
*The Psychology of Computer Programming.*  
John Wiley & Sons, Inc., USA, 1985.
- [40] Andrew Hoffman.  
*Web Application Security: Exploration and Countermeasures for Modern Web Applications.*  
O'Reilly Media, Inc., 2020.

- [41] Leszek A. Maciaszek.  
*Requirements Analysis and System Design.*  
Addison-Wesley Harlow, 3rd edition, 2007.
- [42] Architecture Capability Team.  
*NATO Architecture Framework.*  
NATO, 4th edition, September 2020.
- [43] Nick Rozanski and Eóin Woods.  
*Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.*  
Addison-Wesley, 2nd edition, 2012.
- [44] The Open Group Architecture Forum.  
*The Open Group Architecture Framework Standard.*  
The Open Group, 9.2 edition, 2018.  
<https://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html>.

[45] *ISO/IEC/IEEE 42010:2011.*

ISO, 2011.

[46] *Unified Modeling Language.*

OMG, 2.5.1 edition, December 2017.

<https://www.uml.org/>.