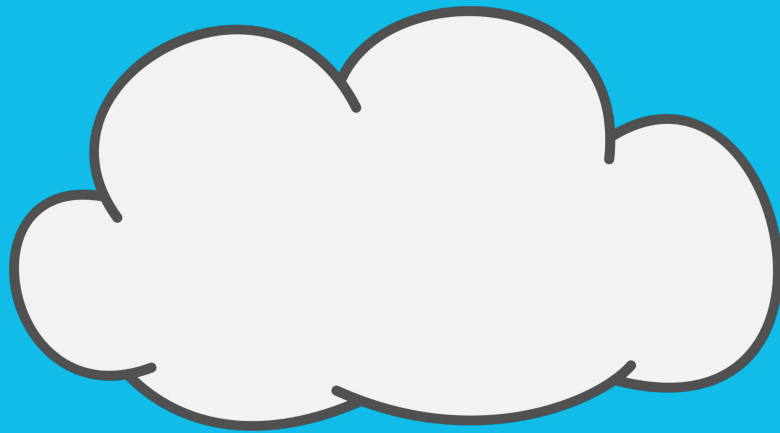


Ben Davis



CI/CD DevOps Presentation  
Feb 2023

# Project Goals

Deploy a website that is:

- Available globally with low latency
- Dynamically scalable in each individual AZ/server
- Able to be updated simultaneously across regions
- Able to automatically deploy new features from a code Repo
- Able to be monitored for performance issues

# Services Used

AWS EC2 Amazon Linux Servers: To host Jenkins, Apache, Maven, and Ansible

Jenkins: To pick up Github Webhook, Package Webapp, and Deploy to Apache Web Server

AWS Security Groups, Firewalls, Generated SSH Keys

Amazon CloudWatch Dashboard to Monitor completed Infrastructure

Ansible: To configure Linux Servers

Terraform: To provision servers and configure proper security and geographic location

Maven: To package Webapp

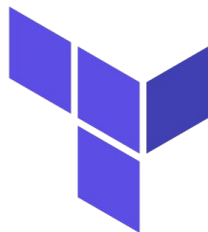
Docker/AWS ECR: To host multiple Web Servers



ANSIBLE



# Jenkins



HashiCorp

# Terraform

# Bring Joy Back to Learning

Learn More about Our Holistic,  
Neuroscience-Based Approach to  
Reversing Dyslexia, ADHD, ADD, and  
Other Learning Challenges.

📞 **(512) 331-0668**

Call today for your  
Free Consultation.

AS SEEN ON



Los Angeles Times

blogtalkradio



The Dallas  
Morning News

AUSTIN  
NEWS  
kxan.com



Books Family Health Center is a local Austin Healthtech Startup

# It's Digital Transformation time.

Due to the COVID pandemic, the business wants to shift from on-premise treatment to digital training for practitioners.

One critical element is the online presence, which is starting to draw attention from other English speaking countries (US, UK, Canada, Australia, NZ)

The increased traffic is causing service outages, and there are excessive lag times in distant countries.

Due to the low capital investment cost, AWS Cloud is a great option for hosting the website (and other components) with increased availability.

# Provision Resources

Using Terraform we will create  
Instances:

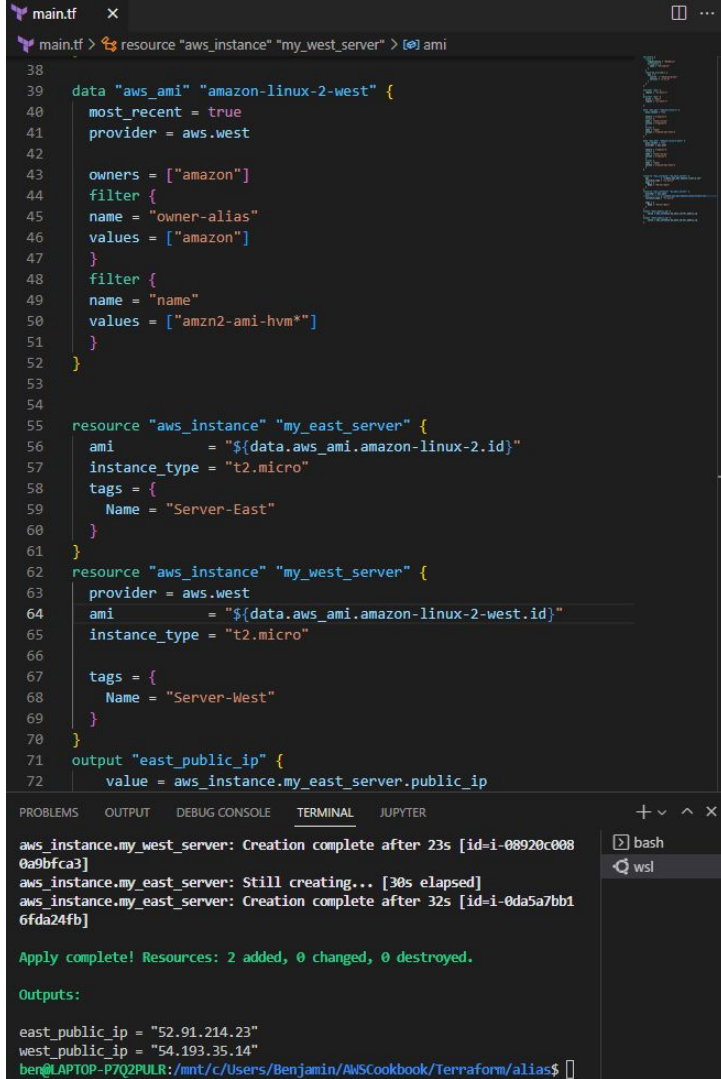
1 to run Ansible

1 to run Jenkins

6 to run Docker and Apache Web  
Server

8 SSH Keys

Make sure all servers are in security  
groups that can talk to each other.  
(Open on port 22)



The image shows a VS Code editor with two files open: `main.tf` and `ami`. The `main.tf` file contains Terraform configuration for two AWS instances. The `ami` file defines an Amazon Linux 2 AMI. The terminal window at the bottom shows the output of a Terraform apply command, indicating that two resources were added successfully.

```
main.tf > resource "aws_instance" "my_west_server" > ami
```

```
38
39 data "aws_ami" "amazon-linux-2-west" {
40     most_recent = true
41     provider   = aws.west
42
43     owners = ["amazon"]
44     filter {
45         name = "owner-alias"
46         values = ["amazon"]
47     }
48     filter {
49         name = "name"
50         values = ["amzn2-ami-hvm*"]
51     }
52 }
53
54
55 resource "aws_instance" "my_east_server" {
56     ami           = "${data.aws_ami.amazon-linux-2.id}"
57     instance_type = "t2.micro"
58     tags = {
59         Name = "Server-East"
60     }
61 }
62
63 resource "aws_instance" "my_west_server" {
64     provider = aws.west
65     ami       = "${data.aws_ami.amazon-linux-2-west.id}"
66     instance_type = "t2.micro"
67
68     tags = {
69         Name = "Server-West"
70     }
71 }
72
73 output "east_public_ip" {
74     value = aws_instance.my_east_server.public_ip
75 }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
```

```
aws_instance.my_west_server: Creation complete after 23s [id=i-08920c0080a9bfca3]
aws_instance.my_east_server: Still creating... [30s elapsed]
aws_instance.my_east_server: Creation complete after 32s [id=i-0da5a7bb16fda24fb]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

east_public_ip = "52.01.214.23"
west_public_ip = "54.193.35.14"
ben@LAPTOP-P7Q2PULR:/mnt/c/Users/Benjamin/AWSCookbook/Terraform/alias$
```

Since we are working in multiple AZs, each server also needs its own load balancer, target group, and listener, and autoscaling group.

Some resources need to be created and attached using AWS and Terraform.

The instances depend on the load balancer and autoscaling group (and security groups) so we can use 'depends\_on' to make sure that the supporting infrastructure is created before we provision the servers.

```
# Filter for instances in AZ1
target_type = "instance"
target_group_targets {
  availability_zone =
"${data.aws_availability_zones.available.names[0]}"
}
```

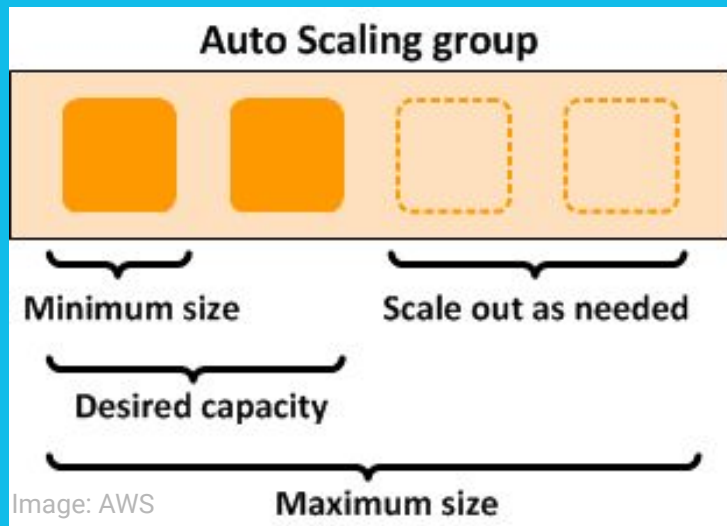
```
# Define the target group for AZ1
resource "aws_lb_target_group" "tg_az1" {
```

```
# Define the listener for AZ1
resource "aws_lb_listener" "listener_az1" {
  load_balancer_arn = aws_lb.lb_az1.arn
}
```

```
# Define the load balancer for AZ1
resource "aws_lb" "lb_az1" {
```

```
aws_launch_configuration.example.id
min_size          = 1
max_size          = 3
desired_capacity  = 1
```

# How does the auto scaling group work?



The auto scaling group can create clones of our server to help deal with increased demand.

When the auto scaling group is created, we can specify how and when we want to create more servers.

In this case, it would be great to scale out based on CPU Utilization. If the initial server starts to get overburdened, we can create more to share the load.

Once traffic is manageable again, we can delete the extra servers to save on cost.



# What does the load balancer do?

The load balancer will automatically distribute traffic across multiple servers within our AZ.

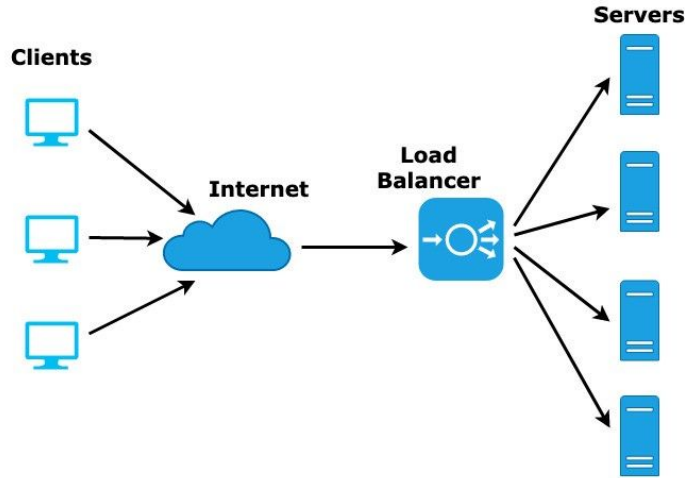


Image: codeburst.com

When the load balancer is created, we can specify where it listens for traffic (listener), where it will send that traffic (target group).

# The instances are now scalable and highly available.

Now that our servers are up and running and supported by various AWS services, it's time to actually install our software on them.

First we will work with our 'Management' node and our 'Deployment' server, which will be in charge of configuring our outward facing web servers.

# Install Ansible

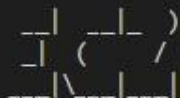
To help us manage the other servers.

```
sudo yum install -y ansible  
(Server 1)
```

# Install Maven

We will need it to run the Jenkins pipeline.

```
sudo amazon-linux-extras install -y maven  
(Server 2)
```



Amazon Linux 2 AMI

```
https://aws.amazon.com/amazon-linux-2/  
4 package(s) needed for security, out of 11 available  
Run "sudo yum update" to apply all updates.  
[ec2-user@ip-172-31-86-202 ~]$
```

# Install Jenkins

The Jenkins server will be the powerhouse of the CI/CD Pipeline.

Jenkins will be connected to our Github repo hosting our Web app.

Jenkins will rely on Maven package manager to package our app and deploy it to the web servers.

## Getting Started

### Getting Started

<input checked="" type="checkbox"/> Folders Plugin	<input checked="" type="checkbox"/> OWASP Markup Formatter	<input checked="" type="checkbox"/> Build Timeout	<input checked="" type="checkbox"/> Credentials Binding Plugin
<input checked="" type="checkbox"/> Timestampers	<input checked="" type="checkbox"/> Workspace Cleanup	<input checked="" type="checkbox"/> Ant	<input type="checkbox"/> Gradle
<input type="checkbox"/> Pipeline	<input type="checkbox"/> GitHub Branch Source Plugin	<input type="checkbox"/> Pipeline: GitHub Groovy Libraries	<input type="checkbox"/> Pipeline: Stage View
<input type="checkbox"/> Git plugin	<input type="checkbox"/> SSH Build Agents	<input type="checkbox"/> Matrix Authorization Strategy	<input type="checkbox"/> PAM Authentication
<input type="checkbox"/> LDAP	<input type="checkbox"/> Email Extension	<input type="checkbox"/> Mailer Plugin	

Folders

OWASP Markup Formatter

Build Timeout

Credentials Binding

Timestampers

\*\* Resource Disposer

Workspace Cleanup

Ant

\*\* - required dependency

Jenkins 2.332.3

# Configuring Jenkins

The source file must be specified

The remote directory to deliver the package must be specified

The build step including Maven packaging must be specified

The proper version and file path of Git and Maven must be specified

# Creating the Jenkins Pipeline

Jenkins must be told which version of Maven to use.

Jenkins must be configured to publish via SSH connection.

The Pipeline must be built step by step.

The pipeline we are using must also be configured to use Github, and it must be pointed toward the appropriate repo which contains the web app source code.

The pipeline must be pointed toward the web servers, and have access.

# How does the Pipeline work?

Once the pipeline is installed, it can be triggered automatically based on the conditions set:

1. A new commit is created in the specified repository.
2. Jenkins gathers the files.
3. Jenkins manages Maven, which packages the files into a .war
4. The package is tested for bugs.
5. The .war package is deployed to the specified Web Servers

The screenshot shows the Jenkins web interface for a pipeline named 'testing\_pipeline\_1'. The left sidebar contains navigation links: Back to Dashboard, Status, Changes, Build Now (highlighted), Configure, Delete Pipeline, Full Stage View, Open Blue Ocean, Rename, and Pipeline Syntax. Below these is the Build History section, showing a list of builds with a search filter and a 'trend' button. The main content area displays the 'Pipeline testing\_pipeline\_1' view, including a 'Recent Changes' section and a 'Stage View' table. The 'Stage View' table shows the progress of three stages: Build, Test, and Deliver. The 'Build' stage is currently running, while 'Test' and 'Deliver' are completed. The 'Test' stage shows a failed build (#1) with a time of 657ms. The 'Deliver' stage shows a failed build (#1) with a time of 300ms. The 'Permalinks' section at the bottom provides links to the last build and the last failed build.

**Jenkins** Search Benjamin Davis log out

Dashboard > testing\_pipeline\_1

Back to Dashboard

Status

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Open Blue Ocean

Rename

Pipeline Syntax

Build History trend

Filter builds...

#2 Oct 23, 2022 9:43 PM CDT

#1 Oct 23, 2022 9:42 PM CDT

### Pipeline testing\_pipeline\_1

Add description

Disable Project

Recent Changes

### Stage View

	Build	Test	Deliver
Average stage times: (Average full run time: ~7s)	1s	519ms	333ms
#2 Oct 24 02:43 No Changes	1s	381ms	367ms
#1 Oct 24 02:42 No Changes	1s	657ms failed	300ms failed

### Permalinks

- Last build (#1), 1 min 36 sec ago
- Last failed build (#1), 1 min 36 sec ago

# Now where are the Web Servers?

Before we create the pipeline, we need live web servers which are capable of hosting our web app package. On our 6 remaining servers we will...

## Install Docker

So that we can scale our web servers.

```
sudo yum install docker  
(Servers 3-8)
```

And we will use Ansible to help speed up installation, as well as automate any future maintenance.



# Automation with Ansible

Installing Docker manually would be fine, except we don't want to do the same process 6 times. It's time to put our Ansible server to work by creating a YAML script which will run commands on our remote machines.

If we create an inventory file containing the 6 IP addresses we are targeting, we can run a script using the following command:

```
ansible-playbook -i inventory.ini docker-install.yml
```

So what should the script do?

The first step is to get Docker running on each server.



```
PLAY RECAP *****
OK= 6
```

docker-install.yml

```
- name: Install Docker on EC2 instance
  hosts: ec2
  become: true
  tasks:
    - name: Install Docker
      yum:
        name: docker
        state: present
    - name: Start Docker service
      service:
        name: docker
        state: started
```

The second script will install Apache Web Server in the container.

Docker will be the foundation of our web servers.

It makes it fast and easy to install Apache,

And has built in support for auto-scaling in the event of traffic spikes.

Each instance in a separate AZ is has the potential to be its own self contained autoscaling group which can dynamically respond to demand.

```
ansible-playbook -i inventory.ini docker-install.yml
```

apache-install.yml

```
- name: Deploy Apache web server in Docker
  container:
    hosts: ec2
    become: true
    tasks:
      - name: Pull Apache Docker image
        docker_image:
          name: httpd:latest
          state: present
      - name: Start Apache Docker container
        docker_container:
          name: my-apache-container
          image: httpd:latest
          state: started
          ports:
            - "80:80"
```

# Double check the IP address. We are live!

## Test Page

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

### If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting `www.example.com`, you should send e-mail to `"webmaster@example.com"`.

### If you are the website administrator:

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the image below on web sites powered by the Apache HTTP Server:



# Now we can finalize and run the pipeline.

The pipeline needs to be configured to deploy in parallel to each of the 6 servers.

We can use a Jenkinsfile script to give it instructions.

The 'stage' and 'parallel' commands will help.

```
...
stage('Deploy') {
    parallel {
        stage('Deploy to web1') {
            steps {
                sh 'scp target/*.war user@web1:~/'
            }
        }
        stage('Deploy to web2') {
            steps {
                sh 'scp target/*.war user@web1:~/'
            }
        }
        stage('Deploy to web3') {
            steps {
                sh 'scp target/*.war user@web1:~/'
            }
        }
    }
}
...
```


All we have to do is commit our Web app to Github,  
the pipeline will be triggered!

```
Benjamin@LAPTOP-P7Q2PULR MINGW64 ~/HTML5 (master)
$ git push
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 158.19 KiB | 15.82 MiB/s, done.
Total 7 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/benjamindavisdc/benjamindavisdc.github.io/
   dcd8109..70f3cb5  master -> master

Benjamin@LAPTOP-P7Q2PULR MINGW64 ~/HTML5 (master)
$ █
```

# Hit refresh on each Apache server. The website should now be live.

info@drphyllisbooks.com 12412 Mossy Bark Trail, Austin, TX 78750 Mon.-Fri. 9am-6pm & Sat. 11am-2pm MEMBER LOGIN f t y


 **Dr. Phyllis Books**  
WHERE HEART & NEUROSCIENCE MEET


GET FREE CONSULTATION!  
**(512) 331-0668**

HOME ABOUT RE-IMAGINING DYSLLEXIA SUCCESS STORIES 1-ON-1 PROGRAMS DIY COURSES SERVICES RESOURCES CONNECT

## Bring Joy Back to Learning

Learn More about Our Holistic,  
Neuroscience-Based Approach to  
Reversing Dyslexia, ADHD, ADD, and  
Other Learning Challenges.

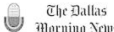
 **(512) 331-0668**  
Call today for your  
Free Consultation.



AS SEEN ON



Los Angeles Times





# And the best part is...

From now on, any time we update the source code in our Repo, the packaging, testing, and deployment will happen again automatically.

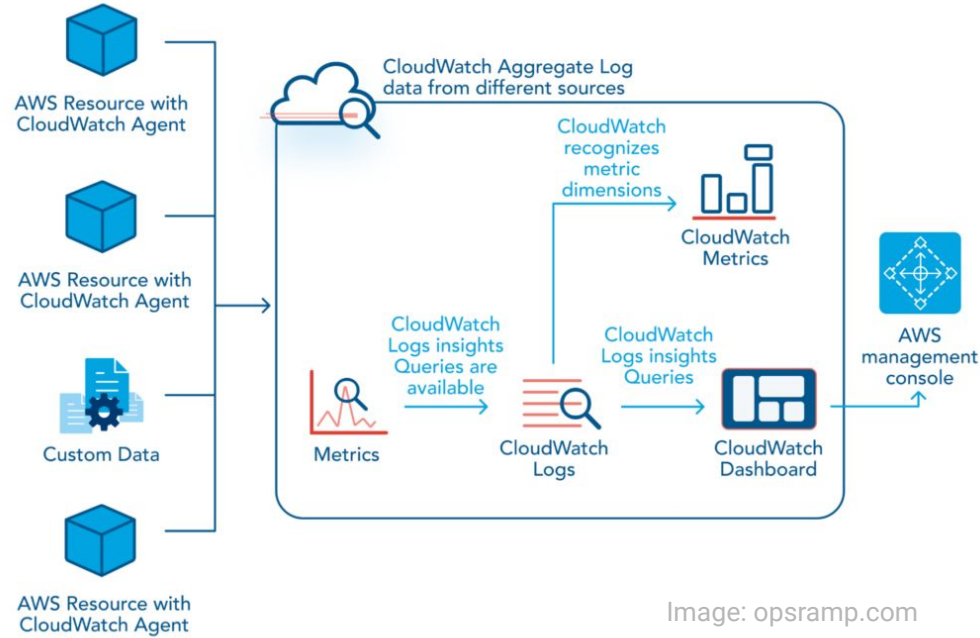
Within ~2000ms, our updated site will be running globally.

Now that the deployment configuration is complete, we have a few more things to do to make sure the instance stays healthy.



# Amazon Cloudwatch will help us monitor our servers.

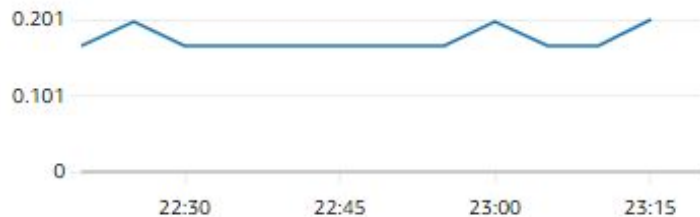
Cloudwatch has a suite of features including Visualized Dashboards that show KPI metrics in real time, alerts that can notify administrators, and detailed logs to help investigate errors.



## Instance: i-0f3bc674e5b8f99b5 (Web Server 1)

CPU utilization (%)

Percent



Status check failed (any) (count)

Count



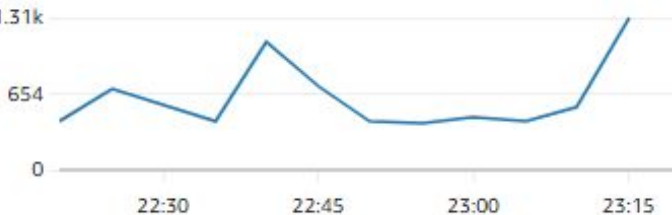
Network in (bytes)

Bytes



Network out (bytes)

Bytes



Monitoring will show us the performance of each server.

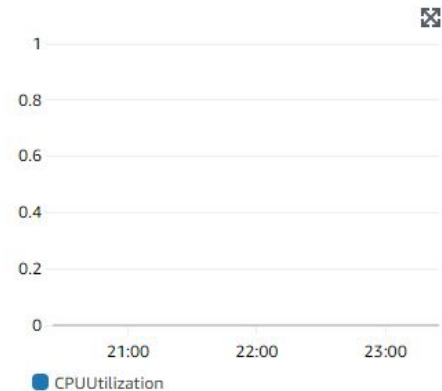
## Specify metric and conditions

**Metric**

Edit

**Graph**

This alarm will trigger when the blue line goes above the red line for 1 datapoints within 5 minutes.



1  
0.8  
0.6  
0.4  
0.2  
0

21:00 22:00 23:00

CPUUtilization

Namespace  
AWS/EC2

Metric name  
CPUUtilization

InstanceId  
i-0d58f5fb319f71296

Instance name  
Jenkins Deploy

Statistic  
Average

Period  
5 minutes

We can configure alarms to warn if things (will soon) go out of control.

# Deployment is complete!

Our servers are operational,

Easily updated via Ansible,

Our web content will automatically be updated with each commit,

We are prepared for traffic spikes,

And we will get an alert if anything goes down.

Benjamin Davis  
DC, SAA-C02, AZ-900

Austin, Tx

[LinkedIn](#)

972 310 8939

benjamin.davis.saa@gmail.com