Building an AWS CloudWatch dashboard using Terraform — 15 minute guide

Antony Melvin
Oct 25, 2019 · 6 min read ★

Terraform is a leading Infrastructure as Code (IaC) tool. This is a short guide on what it is, why you should use it & how to start — in 15 minutes.

IaC means that you define infrastructure in code (usually in a text file) & then predictably generate that infrastructure in the Cloud specified. When you are done with it you can predictably remove the infrastructure created. Terraform is cloud agnostic, so can be applied to any major cloud.

Why Terraform?

Why use Terraform instead of Puppet, Chef, Ansible etc? Well Steve Strutt, IBM CTO, makes a very good case for Terraform, in that Terraform is simply not the same as other leading IaC tools — being an orchestration tool rather than a configuration tool — with his recommendation being to use Terraform to build infrastructure & Puppet, Chef etc to configure, patch and so on. Gruntwork have made similar observations.

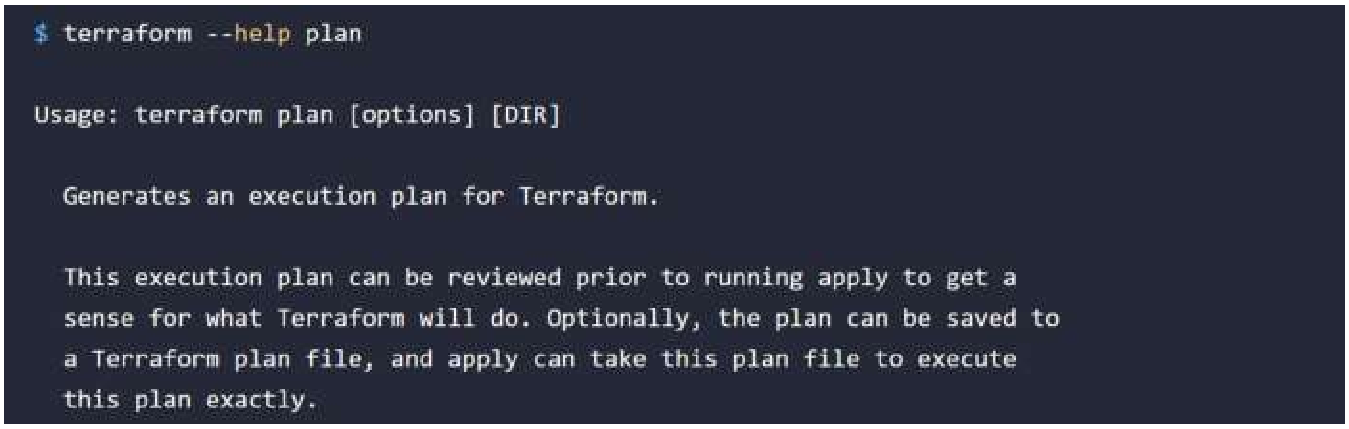
Terraform is immutable so it fits with the "cattle" not "pets" concept that is a solid (apols!) principle of Cloud. Fundamentally losing a piece of infrastructure should be a trivial event that can happen at any time so being able to add or remove infrastructure should be a trivial act — not one whereby the configuration of the infrastructure keeps changing (configuration drift) to the extent that you daren't change stuff.

If you look about you can probably find a more succinct definition that this!

Install Terraform

Download Terraform here and follow the guide here on how to install Terraform on your specific system.

You should be able to see the help options or Terraform version when installed correctly:



```
$ terraform --help plan

Usage: terraform plan [options] [DIR]

  Generates an execution plan for Terraform.

  This execution plan can be reviewed prior to running apply to get a
  sense for what Terraform will do. Optionally, the plan can be saved to
  a Terraform plan file, and apply can take this plan file to execute
  this plan exactly.
```

Building infrastructure on AWS using Terraform

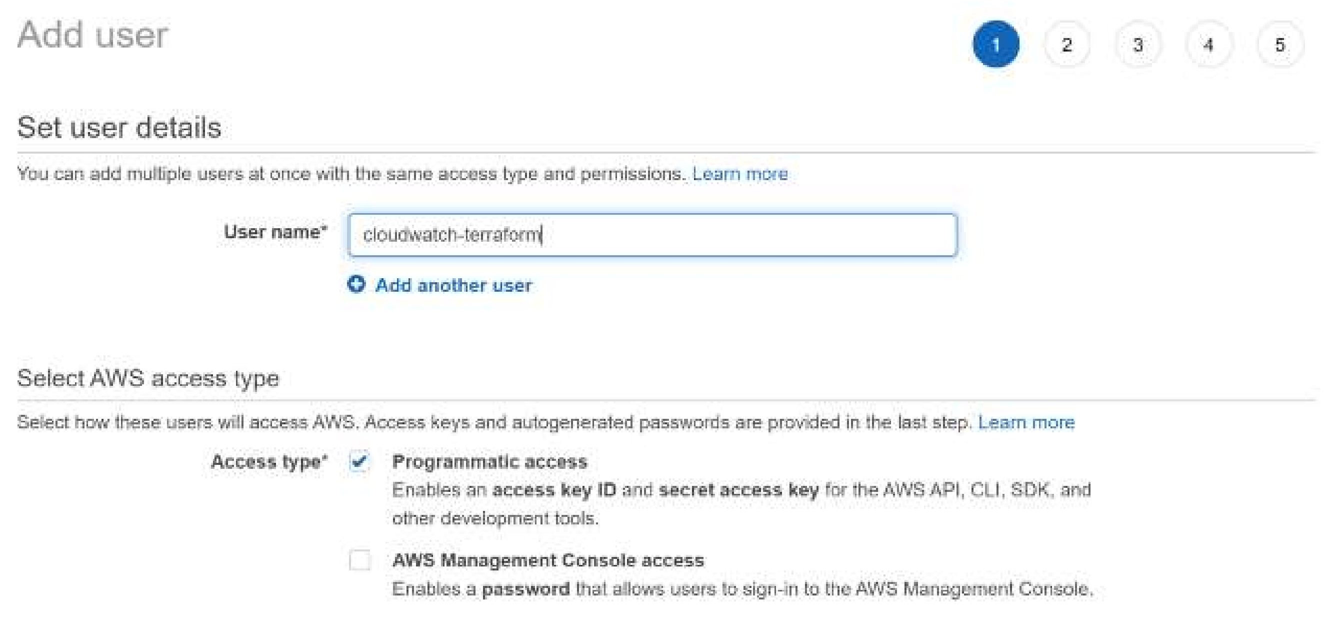
Firstly you'll need access to an AWS account.

Terraform configuration is deployed using the (HCL) HashiCorp Configuration Language.

There are plenty of beginner tutorials for building an EC2 instance in AWS & then destroying it (like here, here & here) & more examples in the Terraform Registry. So instead of rehashing those examples I thought I'd offer a CloudWatch dashboard example, after all CloudWatch is used explicitly by AWS as part of auto-scaling (which looks very much like pre-baked IaC to me).

IAM User

Firstly you'll need to create an AWS IAM user with programmatic access that is able to create the infrastructure you need. Most beginner tutorials suggest that you give full admin rights, but you can be much more granular than that — save the access key & secret key you get at the end for use later:



Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* cloudwatch-terraform

Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* ✓ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

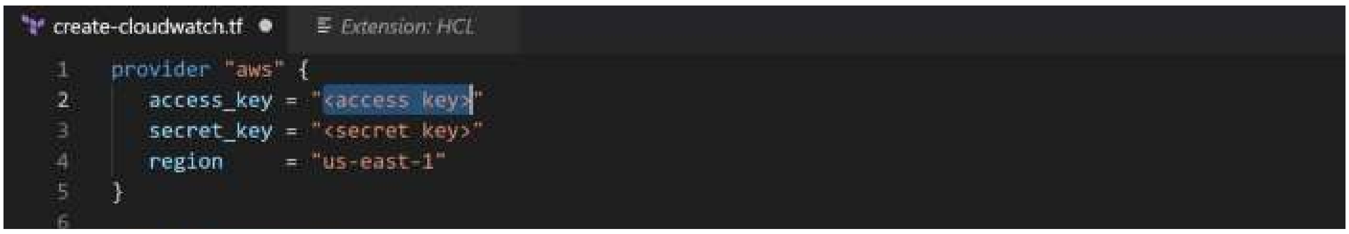
TF File

Create a Terraform file in HCL format with an extention of .tf. If you want a simple start use the first section below to gain access to AWS (provider); then use an official resource example (resource). This will give you a basis to then experiment.

Example TF

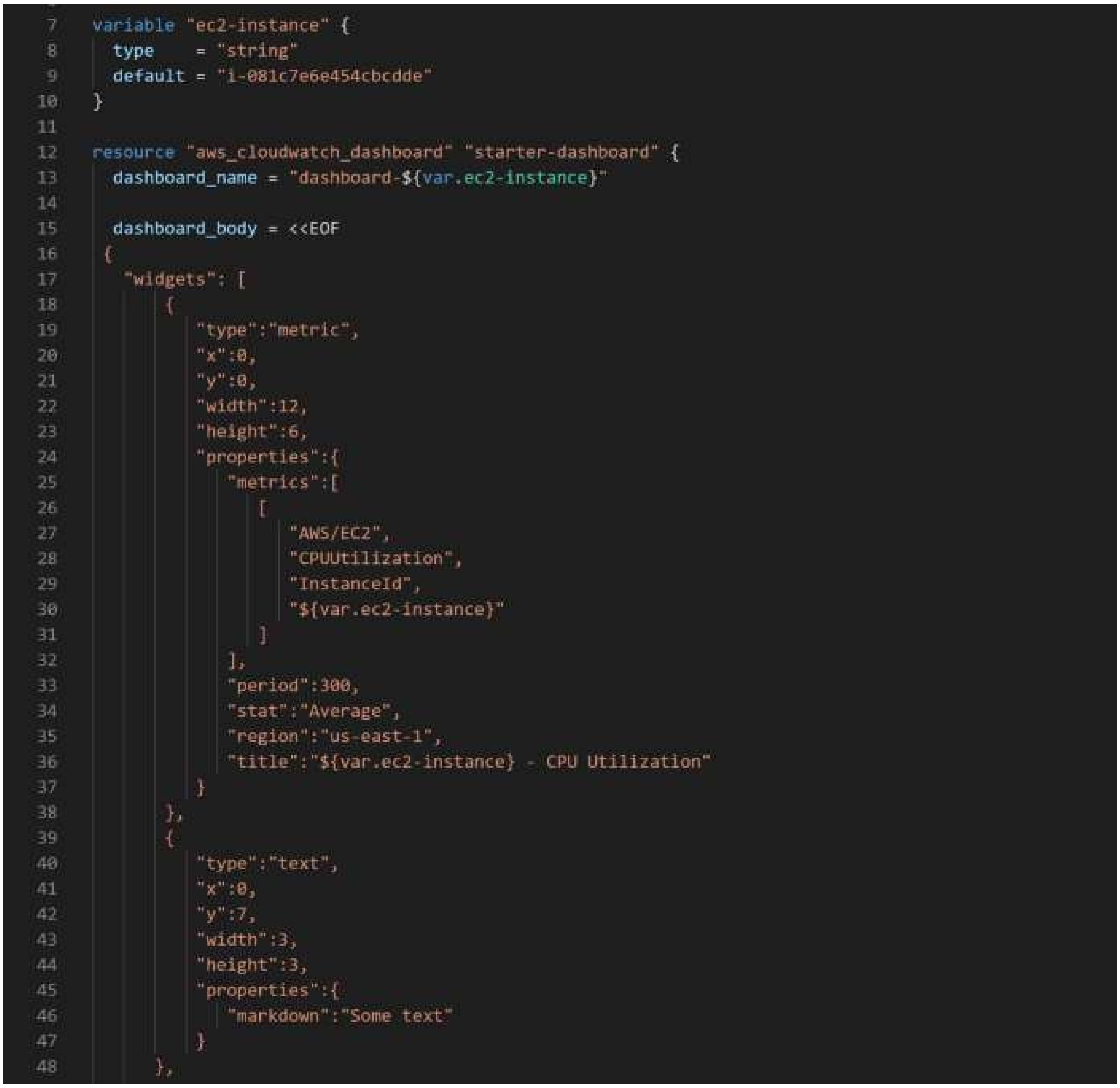
Below is an example for creating a simple CloudWatch Dashboard & Alarm both in AWS. Displayed in VS Code using the official HashiCorp Configuration Language support for Visual Studio Code plug-in.

- First section gives the provider programmatic access to the Cloud (use your own IAM user keys)
- Second section sets up a variable that is used 5 times in the third section
- Third section adds a Cloudwatch dashboard
- Fourth section add a CloudWatch alarm



create-cloudwatch.tf Extension: HCL

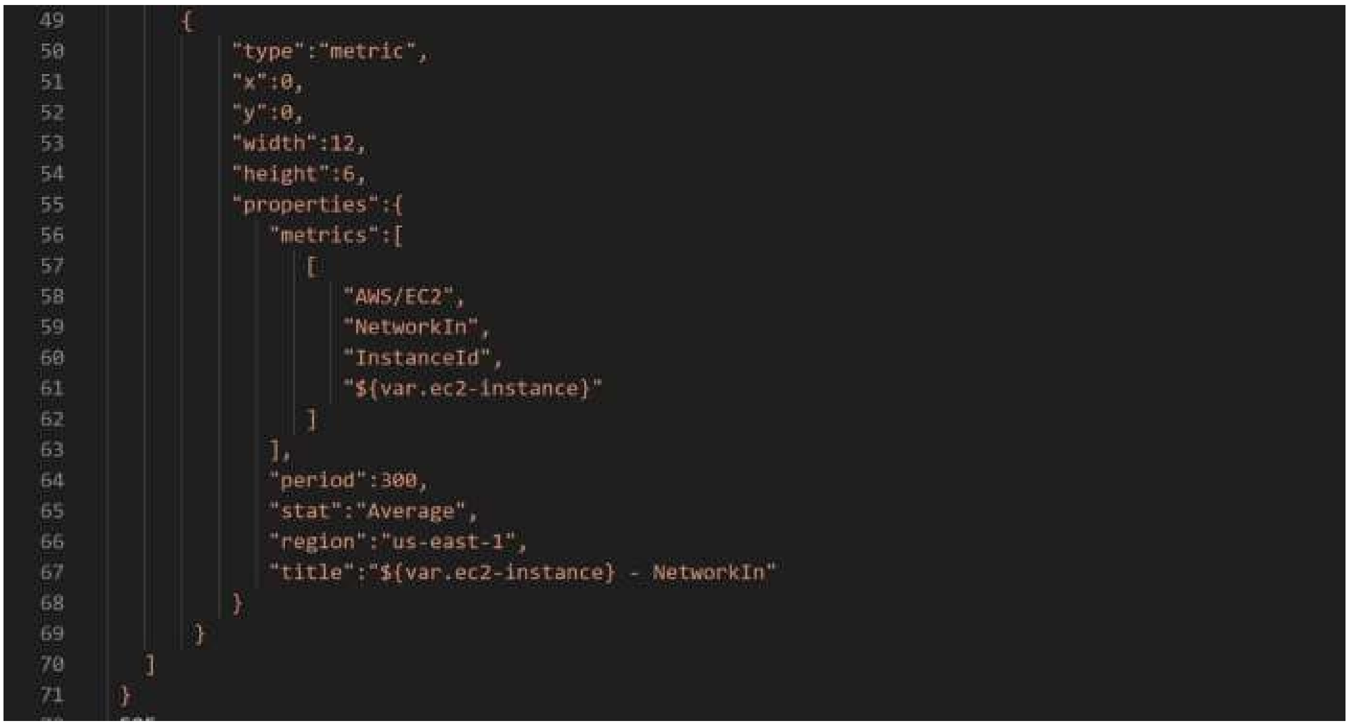
```
provider "aws" {
   access_key = "<access key>"
   secret_key = "<secret key>"
   region     = "us-east-1"
}
```



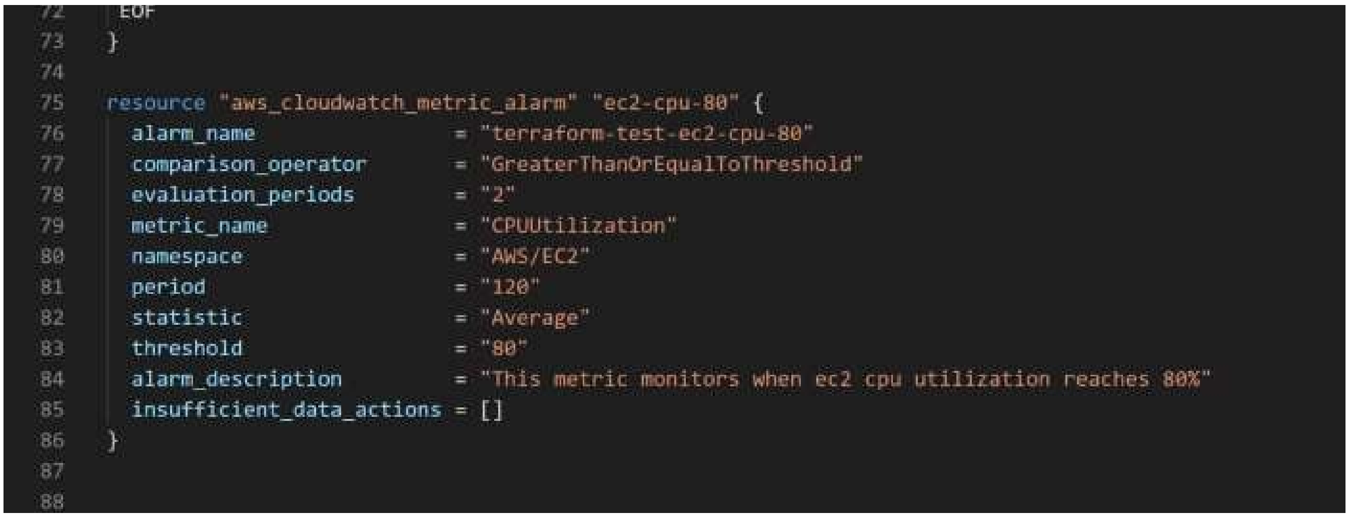
```
variable "ec2-instance" {
  type    = "string"
  default = "i-081c7e6e454cbcdde"
}

resource "aws_cloudwatch_dashboard" "starter-dashboard" {
  dashboard_name = "dashboard-${var.ec2-instance}"

  dashboard_body = <<EOF
 {
   "widgets": [
       {
          "type":"metric",
          "x":0,
          "y":0,
          "width":12,
          "height":6,
          "properties":{
             "metrics":[
                [
                   "AWS/EC2",
                   "CPUUtilization",
                   "InstanceId",
                   "${var.ec2-instance}"
                ]
             ],
             "period":300,
             "stat":"Average",
             "region":"us-east-1",
             "title":"${var.ec2-instance} - CPU Utilization"
          }
       },
       {
          "type":"text",
          "x":0,
          "y":7,
          "width":3,
          "height":3,
          "properties":{
             "markdown":"Some text"
          }
      },
```



```
      {
          "type":"metric",
          "x":0,
          "y":0,
          "width":12,
          "height":6,
          "properties":{
             "metrics":[
                [
                   "AWS/EC2",
                   "NetworkIn",
                   "InstanceId",
                   "${var.ec2-instance}"
                ]
             ],
             "period":300,
             "stat":"Average",
             "region":"us-east-1",
             "title":"${var.ec2-instance} - NetworkIn"
          }
       }
   ]
 }
```



```
  EOF
}

resource "aws_cloudwatch_metric_alarm" "ec2-cpu-80" {
  alarm_name                = "terraform-test-ec2-cpu-80"
  comparison_operator       = "GreaterThanOrEqualToThreshold"
  evaluation_periods        = "2"
  metric_name               = "CPUUtilization"
  namespace                 = "AWS/EC2"
  period                    = "120"
  statistic                 = "Average"
  threshold                 = "80"
  alarm_description         = "This metric monitors when ec2 cpu utilization reaches 80%"
  insufficient_data_actions = []
}
```

Terraform Init

Run terraform init in the same folder as the TF file to initialise your setup:

```
terraform init
```

Terraform Plan

Run terraform plan to build the execution plan:

```
terraform plan
```

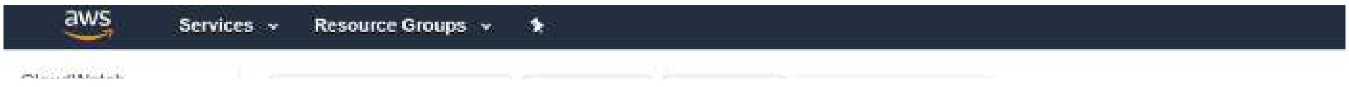
Terraform Apply

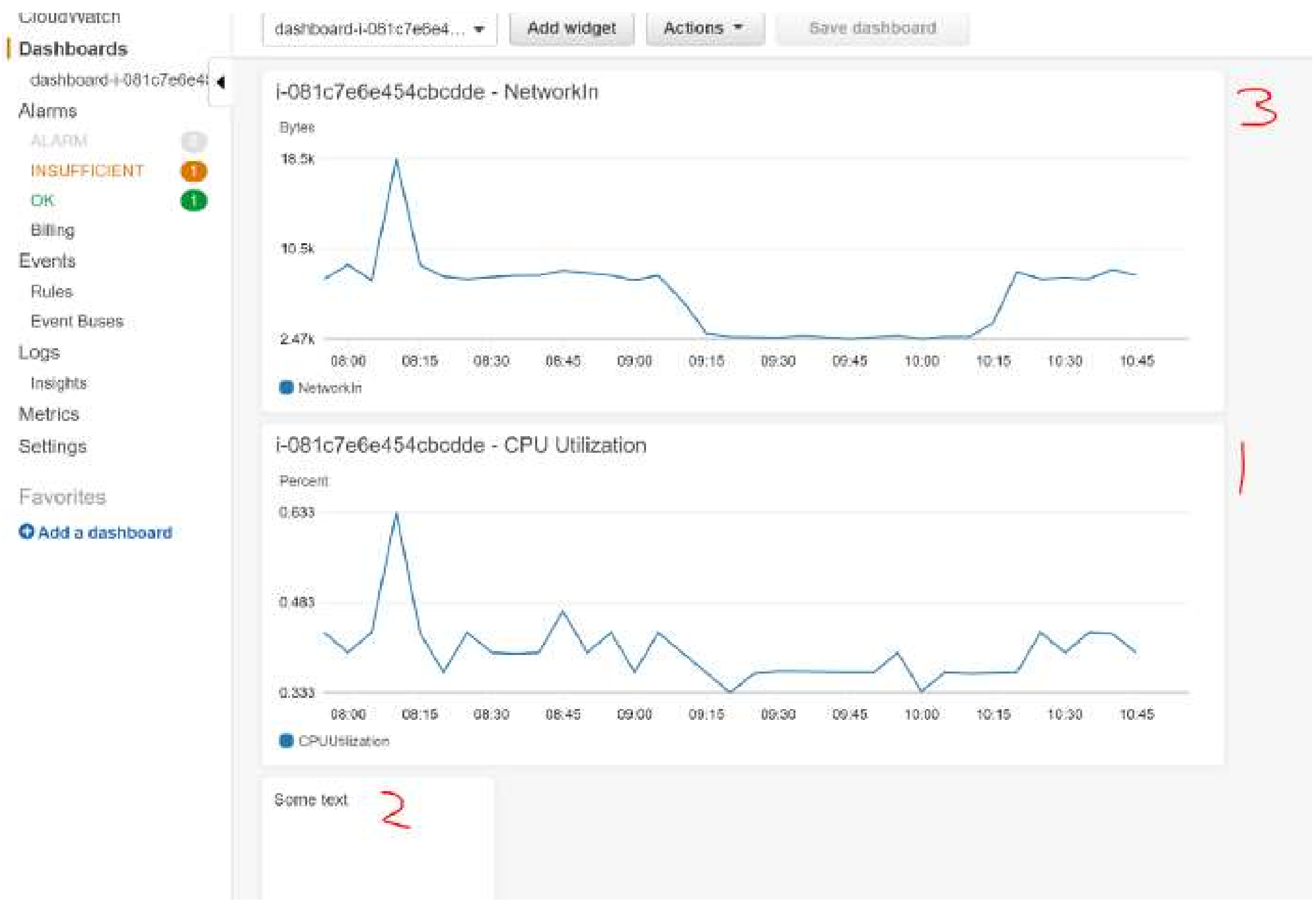
Run terraform apply to apply the plan:

```
terraform apply
```

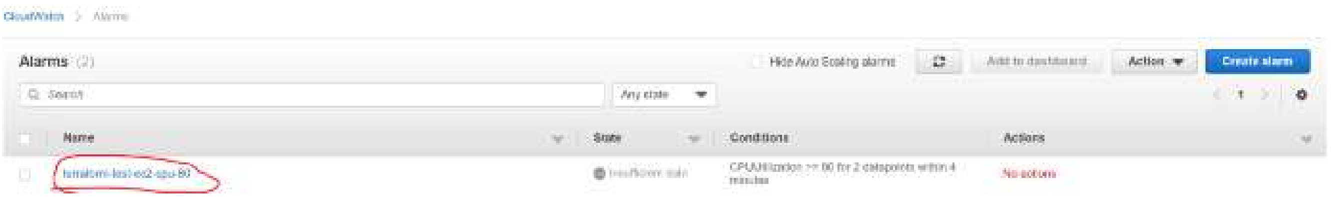
All being well after the terraform apply has completed the following infrastructure will exist:

If you've followed the example code above then you should see a CloudWatch dashboard called dashboard-i-xxxxxxxx with three widgets, you may not see activity on your widgets because I created a variable pointing at a running EC2 instance in my test setup — so you need to do the same by copying an instance id into the default value for the variable (section 2 above):





A CloudWatch alarm that will fire when CPU Utilization hits 80%:



Name	State	Conditions	Actions
terraform-test-ec2-cpu-80	Insufficient data	CPUUtilization >= 80 for 2 datapoints within 4 minutes	No actions

Terraform Destroy

Run terraform destroy to reverse the plan & remove the CloudWatch stuff:

```
terraform destroy
```

So that's Terraform?

So much more to it, but you should now have a working IaC example. In a few short years you'll be an expert!

Other Thoughts

Interpolation

You can make TF files much more generic using interpolation to output the instance id(s) after creation of an EC2 & then input it into CloudWatch terraform apply command using the -var switch (beyond the scope etc & so on…). This would mean you could create a standard dashboard targeting every EC2 instance you launch.

But I'll save all that for another time.

Ref: Terraform a CloudWatch dashboard

Packer

You can also user a tool like Packer to automate your generation of machine images & also bring them into a coding pipeline enabling auto-scaling setup or similar to become trivial.

Baked in Infrastructure as Code: Auto-scaling

Infrastructure in the Cloud, being often virtual from the perspective of a user, naturally lends itself to tools that can manipulate the setup of servers. So a first-class implementation of what I believe is Infrastructure as Code that seems to be baked into every Cloud* is auto-scaling.

Auto-Scaling is where you define server metrics & when these metrics are exceeded the code will be run to scale up (increase the capability of existing servers, usually more RAM/CPU) or scale out (increase the number of server instances).

Auto-scaling in Microsoft Azure | Auto-scaling in AWS | Auto-scaling in Google Cloud

IaC Version Control, CI/CD & a Gotcha

As HCL files are text files it's a simple process to use GIT to maintain version control of Terraform files & as a bonus one of the pre-built .gitignore variants is for Terraform. Be aware that one of the gotchas with Terraform is the flipside of its immutability & declarative nature.

If you APPLY a TF file & then change the TF file & then APPLY it again the declarative aspect means that Terraform will build infrastructure that matches the second declaration — which is good.

But if you don't APPLY the latest version of a TF file, what happens when you come to DESTROY it? Terraform may fail (depending on the conflict) or be hit with a mismatch of what is in the plan & what exists in the cloud. So version control needs a process whereby a DESTROY should only happen against IaC that matches the previous APPLY

— suggesting IAC should really move into a CI/CD pipeline (see here, here & here). But again that is beyond the scope of this article.

Conclusion

Thanks for reading, I think this is enough for a beginner? Terraform is a deep subject with a low barrier to entry — so hopefully this is enough to get you started. Contact me in the usual ways!

AWS Terraform Cloudwatch