

Finalization Phase

Task description

The task is to setup a webpage online using amazon web service and for the infrastructure as a code we are using terraform. First, we have to design the architecture and then deploy it. We are allowed to use the tools we like. but we have certain requirements that we must keep in our mind. The webpage should be highly scalable and auto scale itself if more traffic comes. It can be easily accessible through a public Ip address.

Tools and service used

Amazon web service

AWS is cloud computing service provider that provide cloud-based products like compute, storage, databases, analytics, networking and enterprise application on pay-as-you-go pricing. We are using different aws tools for the successful deployment of our webpage. Aws is one of the most popular cloud service provider with over 39% market share in cloud computing industry.

Elastic cloud compute (EC2)

Amazon Ec2 provide a wide range of instances for variety of use cases. Instance types consists of different CPU's , memory, storage, and networking capacity. EC2 provide virtual server to its user to run different application and web solutions. We are using ec2 instance to launch our html page. Ec2 also provide many more features like CloudWatch , ec2 fleet and elastic Ip address.

Security groups

A security group acts as a firewall that controls the traffic allowed to the resources in our virtual private cloud. We can choose the desired ports and protocol to allow the inbound traffic and the outbound traffic. It will take request from HTTP and SSH ports. Instances can also have multiple security groups for each interface. Since

AWS doesn't deny traffic, each security group will be compounded, allowing access if any of the security groups match for a specific packet.

Autoscaling group

Autoscaling groups contain a collection of ec2 instance that are treated as a logical unit for the purpose of autoscaling and management. It will automatically scale the instance if more visitors come to our webpage. It checks the instance health for replacement. The size of autoscaling group depends on the desired capacity set by us. We can adjust the size as per our demands. It maintains the number of instances by doing a periodic health check on the instance in the group. If the autoscaling group finds an unhealthy instance it automatically terminates the unhealthy instance with a completely new instance.

Launch template

A launch template specifies the configuration information for a instance. We can specify the instance ID, instance type, key pair, security groups and other information to launch a ec2 instance. We will attach our bash script with the launch template, by which every time our instance start it will launch the html page.

Elastic load balancer

Elastic load balancer is a service by AWS, which distribute all the incoming traffic across multiple EC2 instance, containers, and IP addresses. It also monitors the health of the instance and distribute the traffic to the healthy instance. It automatically scales the load balancer capacity in case of change in incoming traffic. There are three different types of load balancer. We are using application load balancer for our project. Application load balancer functions at the application layer. After the load balancer receive a request, it will evaluate the listener to determine the rule of supply. With ec2 instance elastic load balance it improves the scalability and availability of the application. We it using to successfully incoming traffic to different availability zones. This type of Load Balancer is used when there are HTTP and HTTPS traffic routing.

Terraform

Terraform is a infrastructure as code tool that let us define the cloud resources in a configurable file that we can reuse and share it with different cloud providers through API. Terraform is cloud-agnostic and uses a high-level declarative style language called HashiCorp Configuration Language (HCL) for defining infrastructure in a simple configuration file. The core of terraform consists of three steps write, plan and apply. First, we define multiple service and resources than terraform creates a execution plan describing the infrastructure it will create and finally on approval it perform the operation in correct order. We are using the terraform for our project. I made a completely separate file for all the tools, it helps us to understand the code better and in finding potential error. Terraform also have many features that's why I am using it over AWS CloudFormation like it makes the code reusable and we deploy resource faster. Terraform also gives us the opportunity to create and destroy all the resources in just one simple click. Terraform has many notable benefits it can be deployed on multiple cloud providers. Terraform supports built-in functions that can be called and used within your code. The core strength of Terraform is its ability to manage and stick together with huge number of providers. This allows it to act as a bridge between Clouds, SaaS services, network devices and local applications.

The main idea

My main concept is very clear that we are using terraform to implement all our infrastructure. I am using PyCharm as a code editor for writing the terraform code. I have created a complete separate file for every tool I use like elastic load balancer and autoscaling groups. It will help me to understand everything in precise way and help us to write the code in better way. The main file contains information about the provider and the access key to our account. I have created a bash script that will take run the html file every time our instance runs. the autoscaling instance will increase the number of instance when more traffic comes to our website. So, the maximum number of instances is 4 and the desired capacity is one. The health of our instances is automatically checked in every 300 seconds by this if any is not working properly than the autoscaling group will replace it with new one. We have also created the auto scaling policy by which autoscaling will create a new instance whenever one is required. At this stage every service that we are using it almost for free. Ec2 instance comes under 1-year free trial. So, we have spent less than a dollar for all the services and tools that we I am using now.

Some visuals of our services running on aws

Elastic cloud compute (Ec2)

The screenshot displays the AWS Management Console interface for EC2 instances. At the top, there's a header for 'Instances (1/1)' with filters set to 'Instance state = running'. Below this, a table lists the instance 'Myinstance' with ID 'i-074cb6ffc8f956a1b', state 'Running', type 't2.micro', and status 'Initializing'. The instance is located in the 'us-east-2b' availability zone. Below the table, a detailed view for 'Instance: i-074cb6ffc8f956a1b (Myinstance)' is shown, including tabs for Details, Security, Networking, Storage, Status checks, Monitoring, and Tags. The 'Details' tab is active, showing the instance summary with fields like Instance ID, IPv6 address, Hostname type, Public IPv4 address (3.138.142.169), Private IPv4 addresses (172.31.25.213), and Public IPv4 DNS (ec2-3-138-142-169.us-east-2.compute.amazonaws.com).

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Myinstance	i-074cb6ffc8f956a1b	Running	t2.micro	Initializing	No alarms	us-east-2b

Instance: i-074cb6ffc8f956a1b (Myinstance)

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

▼ Instance summary Info

Instance ID i-074cb6ffc8f956a1b (Myinstance)	Public IPv4 address 3.138.142.169 open address	Private IPv4 addresses 172.31.25.213
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-3-138-142-169.us-east-2.compute.amazonaws.com open address
Hostname type IP name: ip-172-31-25-213.us-east-	Private IP DNS name (IPv4 only) ip-172-31-25-213.us-east-2.compute.internal	

Aws provide two different ip address and DNS for the instance two public and two private. It also automatically checks the status of our instance in a particular time period.

Elastic load balancer

The screenshot shows the AWS Management Console for Elastic Load Balancing. The header indicates 'EC2 > Load balancers'. Below, there's a section for 'Load balancers (1/1)' with a description: 'Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.' There are buttons for 'Create load balancer' and 'Actions'. A search bar is present with the text 'Find resources by attribute or tag'. Below this, a table lists the load balancer 'elastic-load-balancer' with columns for Name, DNS name, State, VPC ID, and Availability Zones. The 'elastic-load-balancer' is shown as 'Active' in the 'us-east-2a' and 'us-east-2b' availability zones. Below the table, a detailed view for 'Load balancer: elastic-load-balancer' is shown, including tabs for Details, Security, Networking, Storage, Status checks, Monitoring, and Tags. The 'Details' tab is active, showing the load balancer summary with fields like Load balancer type (Application), Scheme (Internet-facing), Status (Active), Hosted zone (Z3AADJGX6KTTL2), VPC (vpc-0ef05830c36d35c9a), Availability Zones (subnet-06cbb291201d2be26, subnet-063fcd5d531b9d972), IP address type (IPv4), and Date created (July 6, 2023, 09:01 (UTC+02:00)).

EC2 > Load balancers

Load balancers (1/1)
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

[Create load balancer](#)

Find resources by attribute or tag

Name	DNS name	State	VPC ID	Availability Zones
elastic-load-balancer		Active	vpc-0ef05830c36d35c9a	subnet-06cbb291201d2be26 us-east-2a (use2-az1) subnet-063fcd5d531b9d972 us-east-2b (use2-az2)

Load balancer: elastic-load-balancer

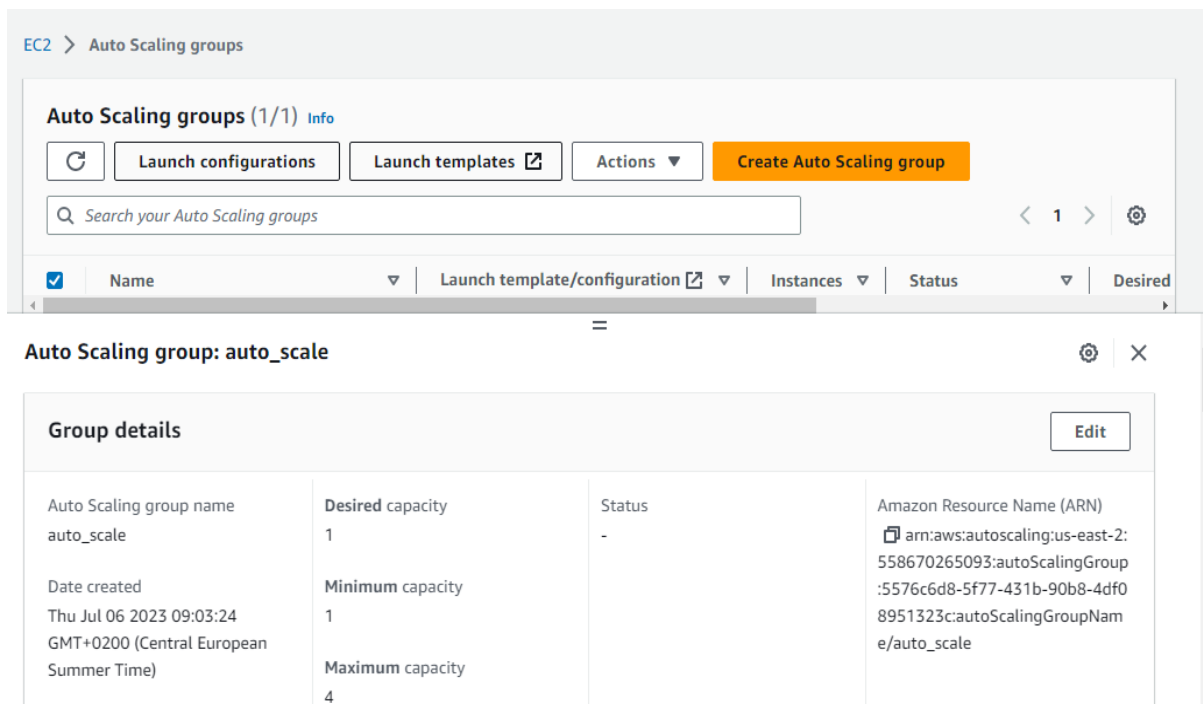
Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Load balancer summary

Load balancer type Application	Status Active	VPC vpc-0ef05830c36d35c9a	IP address type IPv4
Scheme Internet-facing	Hosted zone Z3AADJGX6KTTL2	Availability Zones subnet-06cbb291201d2be26 us-east-2a (use2-az1) subnet-063fcd5d531b9d972 us-east-2b (use2-az2)	Date created July 6, 2023, 09:01 (UTC+02:00)

As you can see we have selected the application load balancer for our project with two different availability zones in east region of USA. It will direct the traffic to healthy instances for better performance. we are using the availability zones in one region it will help in significantly decrease the latency between our instances.

Autoscaling group



The screenshot shows the AWS Management Console interface for the 'Auto Scaling groups' section. The top navigation bar indicates 'EC2 > Auto Scaling groups'. Below this, there's a header for 'Auto Scaling groups (1/1)' with an 'Info' link. A toolbar contains buttons for 'Launch configurations', 'Launch templates', 'Actions', and a prominent orange 'Create Auto Scaling group' button. A search bar is present with the placeholder text 'Search your Auto Scaling groups'. Below the search bar, a table lists the Auto Scaling groups. The first group, 'auto_scale', is selected. The table has columns for 'Name', 'Launch template/configuration', 'Instances', 'Status', and 'Desired'. Below the table, a detailed view for the 'auto_scale' group is shown. It includes a 'Group details' section with an 'Edit' button. The details are organized into four columns: 'Auto Scaling group name' (auto_scale), 'Desired capacity' (1), 'Status' (-), and 'Amazon Resource Name (ARN)' (arn:aws:autoscaling:us-east-2:558670265093:autoScalingGroup:5576c6d8-5f77-431b-90b8-4df08951323:autoScalingGroupName/auto_scale). Other details include 'Date created' (Thu Jul 06 2023 09:03:24 GMT+0200 (Central European Summer Time)) and 'Minimum capacity' (1). The 'Maximum capacity' is listed as 4.

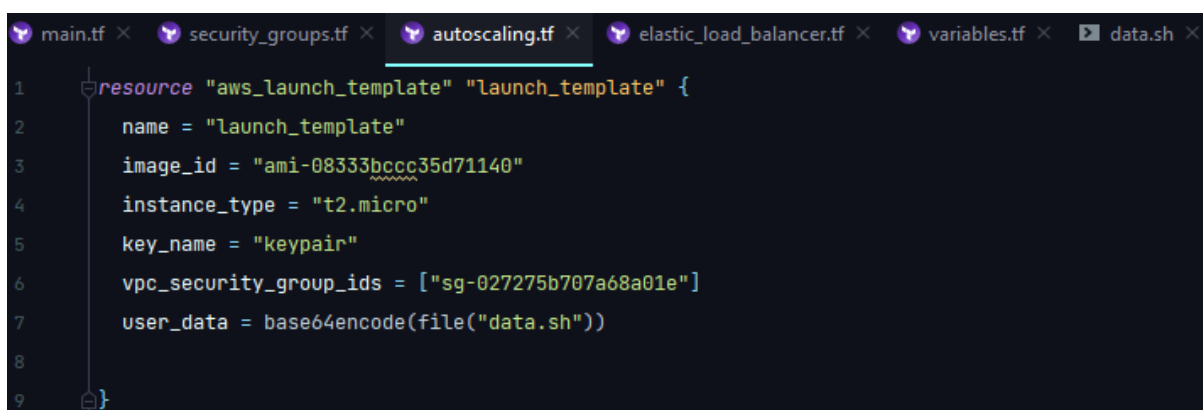
Name	Launch template/configuration	Instances	Status	Desired
auto_scale				

Auto Scaling group: auto_scale

Group details Edit

Auto Scaling group name auto_scale	Desired capacity 1	Status -	Amazon Resource Name (ARN) arn:aws:autoscaling:us-east-2:558670265093:autoScalingGroup:5576c6d8-5f77-431b-90b8-4df08951323:autoScalingGroupName/auto_scale
Date created Thu Jul 06 2023 09:03:24 GMT+0200 (Central European Summer Time)	Minimum capacity 1		
	Maximum capacity 4		

As our autoscaling group clearly meet our desired functionality with desired capacity of 1 instance and maximum capacity of 4 instances. The autoscaling group will check the instance health in every 300 seconds. I have set up the maximum CPU utilization to 70% if CPU utilization increase more than that autoscaling group will automatically add a new instance to our system. Autoscaling group is connected with elastic load balancer by which the traffic is automatically divided into different instances without an additional step.



The screenshot shows a Terraform configuration file named 'autoscaling.tf'. The file is open in a code editor with several tabs at the top: 'main.tf', 'security_groups.tf', 'autoscaling.tf' (active), 'elastic_load_balancer.tf', 'variables.tf', and 'data.sh'. The configuration defines an 'aws_launch_template' resource named 'launch_template'. The configuration includes the following attributes: 'name' is 'launch_template', 'image_id' is 'ami-08333bccc35d71140', 'instance_type' is 't2.micro', 'key_name' is 'keypair', 'vpc_security_group_ids' is a list containing 'sg-027275b707a68a01e', and 'user_data' is the base64 encoded content of the 'data.sh' file. The configuration is enclosed in curly braces and uses the 'resource' keyword.

```
1 resource "aws_launch_template" "launch_template" {
2   name = "launch_template"
3   image_id = "ami-08333bccc35d71140"
4   instance_type = "t2.micro"
5   key_name = "keypair"
6   vpc_security_group_ids = ["sg-027275b707a68a01e"]
7   user_data = base64encode(file("data.sh"))
8 }
9
```

Launch template

The launch template contains the information about how to create the instance. We have to also define other parameters like we have while creating the ec2 instance. We can also modify the user data as per our needs. In our case I have connect the launch template with my GitHub account where the html file data is stored.

```
main.tf x security_groups.tf x autoscaling.tf x elastic_load_balancer.tf x variables.tf x data.sh x
1 resource "aws_launch_template" "launch_template" {
2     name = "launch_template"
3     image_id = "ami-08333bccc35d71140"
4     instance_type = "t2.micro"
5     key_name = "keypair"
6     vpc_security_group_ids = ["sg-027275b707a68a01e"]
7     user_data = base64encode(file("data.sh"))
8
9 }
```

EC2 > Launch templates

Launch templates (1/4) Info Refresh Actions Create launch template

Launch template ID	Launch template name	Default version	La
lt-040a3489ef2f17474	Mytemplate	1	1

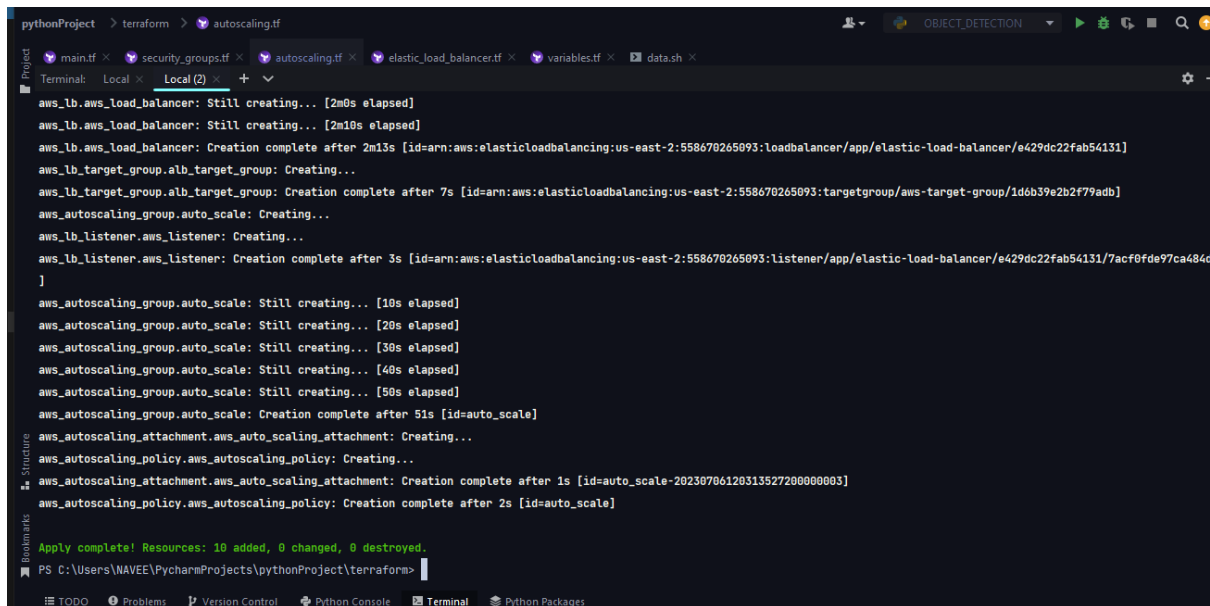
launch_template (lt-06af9cf427597ad40) Settings Close

Launch template details Actions Delete template

Launch template ID lt-06af9cf427597ad40	Launch template name launch_template	Default version 1	Owner arn:aws:iam::558670265093:us er/admin
--	---	----------------------	---

[Details](#) | [Versions](#) | [Template tags](#)

Terraform code after final approval



```
pythonProject > terraform > autoscaling.tf
aws_lb.aws_load_balancer: Still creating... [2m0s elapsed]
aws_lb.aws_load_balancer: Still creating... [2m10s elapsed]
aws_lb.aws_load_balancer: Creation complete after 2m13s [id=arn:aws:elasticloadbalancing:us-east-2:558670265093:loadbalancer/app/elastic-load-balancer/e429dc22fab54131]
aws_lb_target_group.alb_target_group: Creating...
aws_lb_target_group.alb_target_group: Creation complete after 7s [id=arn:aws:elasticloadbalancing:us-east-2:558670265093:targetgroup/aws-target-group/1d6b39e2b2f79adb]
aws_autoscaling_group.auto_scale: Creating...
aws_lb_listener.aws_listener: Creating...
aws_lb_listener.aws_listener: Creation complete after 3s [id=arn:aws:elasticloadbalancing:us-east-2:558670265093:listener/app/elastic-load-balancer/e429dc22fab54131/7acf0fde97ca484d]
aws_autoscaling_group.auto_scale: Still creating... [10s elapsed]
aws_autoscaling_group.auto_scale: Still creating... [20s elapsed]
aws_autoscaling_group.auto_scale: Still creating... [30s elapsed]
aws_autoscaling_group.auto_scale: Still creating... [40s elapsed]
aws_autoscaling_group.auto_scale: Still creating... [50s elapsed]
aws_autoscaling_group.auto_scale: Creation complete after 51s [id=auto_scale]
aws_autoscaling_attachment.aws_auto_scaling_attachment: Creating...
aws_autoscaling_policy.aws_autoscaling_policy: Creating...
aws_autoscaling_attachment.aws_auto_scaling_attachment: Creation complete after 1s [id=auto_scale-20230706120313527200000003]
aws_autoscaling_policy.aws_autoscaling_policy: Creation complete after 2s [id=auto_scale]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
PS C:\Users\NAVEE\PycharmProjects\pythonProject\terraform>
```

After the final approval in the terminal. Terraform starts to create all the planned resources and it also gives step by step visuals of creation with the amount of time spend into creation. It also gives information about the error with any resource, but this will not impact the rest of the program. It also takes an additional approval in the form of yes or no at the time of final approval.

Current status

I have checked all the functionality of the project very well and I think everything is working in the right way. I have checked the webpage with increase in CPU utilization up to 80%. This way we can check that our autoscaling group and elastic load balancer is working fine. I am thinking to add a SSL certificate to our webpage It basically adds another layer of security to our system. But this is out of scope of this task. I am still working on finding any additional bug if possible.

Final result

After testing all the services. I believe that everything works fine as per our need for this project. The infrastructure can be run from any platform with terraform installed on it. I have encountered many problems while testing my terraform code in autoscaling groups and elastic load balancer. But now everything works fine. The webpage can be accessed through the HTTP and SSH protocol from any browser. As you can see it can be accessed with the Ip address. The project gives me the opportunity to work with cloud which itself is a great tool to run different operating

system and application online at very cheaper price as compared to on-premises infrastructure. Moreover, it has tremendous features as compared to any other cloud computing modal at very cheap price.

