Click to verify

Photo by Author Irfan DanishIn my previous article Building a Secure AWS VPC with Terraform: Subnets, Internet Gateways, and More Part-I and Part-II , we explored the process of provisioning a secure Virtual Private Cloud (VPC) infrastructure on AWS. We discussed various components like subnets, security groups, and network ACLs to ensure a robust and protected environment. Building upon that foundation, today we will delve into the concept of an EC2 bastion host.In todays interconnected world, ensuring the security of our cloud infrastructure is paramount. As organizations increasingly adopt cloud technologies, it becomes crucial to implement robust security measures to safeguard sensitive data and systems. One such security measure is the use of an bastion host an essential component of a secure infrastructure setup. In this article, we will explore the concept of an EC2 bastion host, its significance, the benefits it offers in enhancing security within your AWS environment and walk through the step-by-step process of setting it up. So, if youre ready to strengthen the security of your infrastructure and manage your resources more effectively, lets dive into the world of EC2 bastion hosts.Demystifying the Bastion Host:A bastion host, also known as a jump box or a jump server, acts as a fortified gateway between external networks (e.g., the internet) and internal private networks (private subnets). While this isolation adds an extra layer of security, it can pose challenges when administrators or developers require remote access for maintenance, debugging, or troubleshooting purposes. This is where a bastion host comes into play, it serves as a secure and controlled access point, allowing authorized users to connect to resources within the private network from remote locations. By acting as a single entry point, the bastion host minimizes the exposure of your internal network, reducing the attack surface and strengthening the overall security posture.Benefits of Using an EC2 Bastion Host:Enhanced Security: The bastion host acts as a gatekeeper, granting access to specific users or IP addresses while enforcing strong authentication and secure protocols. It helps protect your private instances from direct exposure to the internet, reducing the risk of unauthorized access and potential security breaches.Centralized Access Control: With an EC2 bastion host in place, you can enforce fine-grained access control policies, limiting the number of users who have direct access to the private instances. This centralized approach ensures better governance and auditing capabilities, making it easier to track and monitor access to critical resources.Simplified Networking: By utilizing an EC2 bastion host, you can streamline your network architecture. Rather than exposing multiple instances to the public internet, you consolidate remote access through a single entry point, reducing the complexity of managing and securing multiple access points.Monitoring and Auditing: An EC2 bastion host provides a centralized location to collect logs, monitor user activity, and track SSH sessions. This facilitates better visibility into who is accessing your resources and enables auditing capabilities to meet compliance requirements.To continue our journey towards building a secure and well-managed AWS infrastructure using Terraform, we will now focus on provisioning an EC2 bastion host. As a prerequisite, we will be utilizing the VPC infrastructure that we have set up in our previous article, Building a Secure AWS VPC with Terraform: Subnets, Internet Gateways, and More Part II. In case you missed it, we highly recommend giving it a read as it lays the groundwork for the bastion host setup.All of the code for this article can be found on my GitHub Terraform RepositoryVariables for EC2 Bastion Host# EC2 Bastion Host variablesvariable "ec2-bastion-public-key-path" { type = string}variable "ec2-bastion-private-key-path" { type = string}variable "ec2-bastion-ingress-ip-1" { type = string}2. terraform.tfvars file.## EC2 Bastion Host Variablesec2-bastion-public-key-path = "../secrets/ec2-bastion-key-pair.pub"ec2-bastion-private-key-path = "../secrets/ec2-bastion-key-pair.pem"ec2-bastion-ingress-ip-1 = "0.0.0.0/0"Photo by Markus Spiske on UnsplashCaution! In the terraform.tfvars file we have temporarily setup the value of ec2-bastion-ingress-ip-1 (incoming traffic range) to 0.0.0.0/0 for demonstration purposes. This configuration allows unrestricted access to the EC2 bastion host from the public internet using your SSH key. However, its important to note that this approach is highly insecure. In a real-world scenario, it is strongly recommended to restrict incoming traffic only to trusted sources, such as your organizations VPN or the CIDR range of your organizations network. By implementing these restrictions, you ensure a much higher level of security for your EC2 bastion host and protect it from unauthorized access.Generating SSH Key Pair:Our next step is to generate SSH Key pair for our EC2 Bastion host and create AWS SSH Key Pair resource as well. There are two ways you can create and manage SSH Key Pair for EC2 Bastion Host, we demonstrate both:Using Terraform tls_private_key resource:## Generate PEM (and OpenSSH) formatted private key.resource "tls_private_key" "ec2-bastion-host-key-pair" { algorithm = "RSA" rsa_bits = 4096}## Create the file for Public Keyresource "local_file" "ec2-bastion-host-public-key" { depends_on = [tls_private_key.ec2-bastion-host-key-pair] content = tls_private_key.ec2-bastion-host-key-pair.public_key_openssh filename = var.ec2-bastion-public-key-path}## Create the sensitive file for Private Keyresource "local_sensitive_file" "ec2-bastion-host-private-key" { depends_on = [tls_private_key.ec2-bastion-host-key-pair] content = tls_private_key.ec2-bastion-host-key-pair.private_key_pem filename = var.ec2-bastion-private-key-path file_permission = "0600"}## AWS SSH Key Pairresource "aws_key_pair" "ec2-bastion-host-key-pair" { depends_on = [local_file.ec2-bastion-host-public-key] key_name = "${var.project}-ec2-bastion-host-key-pair-${var.environment}" public_key = tls_private_key.ec2-bastion-host-key-pair.public_key_openssh}Terraform provides the tls_private_key resource, which allows you to generate an SSH key pair within your Terraform configuration. This option is convenient if you prefer managing the key generation process directly in your Terraform code. You can define the key pair parameters, such as the algorithm, length, and format, within the resource block.You can see from above code snippet we are using tls_private_key resource to generate the SSH private and public keys, and then we use local_file and local_sensitive_file resources to store and manage the generated private key securely. The local_file resource allows you to save the private key to a local file on your machine, which can be useful for accessing it outside of Terraform. However, since the private key contains sensitive information, such as authentication credentials, it's crucial to protect it. Here, the local_sensitive_file resource comes into play by encrypting the private key and storing it securely.By using both local_file and local_sensitive_file resources, you can ensure that the private key is safely stored and easily accessible when needed, striking a balance between convenience and security in managing your SSH key pair.First we need to create the SSH Key Pair using following command:ssh-keygen -t rsa -C "you.email@example.com" -b 4096It will prompt you to enter the complete path to the file in which to save the key enter the path e.g path-to-repo/terraform-iac/aws/infrastructure/secrets/ec2-bastion-key-pair. Then it will ask for password enter a secure password for key. Then change the permissions of the SSH private key so that only your user can access the key by running the following command:chmod 600 path-to-repo/terraform-iac/aws/infrastructure/secrets/ec2-bastion-key-pairOnce the SSH Key Pair is generated we will write our AWS Key Pair resource as follows:## AWS SSH Key Pairresource "aws_key_pair" "ec2-bastion-host-key-pair" { key_name = "${var.project}-ec2-bastion-host-key-pair-${var.environment}" public_key = file(var.ec2-bastion-public-key-path)}Networking (Security Group & Elastic IP):Now we will create a Security Group to control the inbound and outbound traffic for EC2 bastion host instance:resource "aws_security_group" "ec2-bastion-sg" { description = "EC2 Bastion Host Security Group" name = "${var.project}-ec2-bastion-sg-${var.environment}" vpc_id = aws_vpc.kodetronix-vpc.id ingress { from_port = 22 to_port = 22 protocol = "tcp" cidr_blocks = [var.ec2-bastion-ingress-ip-1] description = "Open to Public Internet" } egress { from_port = 0 to_port = 0 protocol = "-1" ipv6_cidr_blocks = ["::/0"] description = "IPv6 route Open to Public Internet" } egress { from_port = 0 to_port = 0 protocol = "-1" cidr_blocks = ["0.0.0.0/0"] description = "IPv4 route Open to Public Internet" }}In the above terraform resource block we create a security group in our VPC (that we have created in our previous article). For ingress traffic, the security group allows TCP traffic on port 22 (SSH) from a specific IP address defined by the variable var.ec2-bastion-ingress-ip-1, you can also add multiple IP addresses as well. This configuration allows SSH access to the EC2 bastion host from the specified IP address, typically used for secure remote administration.Regarding egress traffic, the security group allows all traffic (protocol -1) to be sent out to the public internet. This includes both IPv4 (cidr_blocks = ["0.0.0.0/0"]) and IPv6 (ipv6_cidr_blocks = ["::/0"]) traffic. These rules ensure that the EC2 bastion host can communicate with external resources if necessary.Next we will create Elastic IP for our EC2 bastion host. Using an Elastic IP (EIP) for a bastion host offers several benefits and is considered a best practice in secure infrastructure design. By associating an EIP with the bastion host, you ensure that its public IP address remains consistent even if the instance is stopped, restarted or replaced. This provides stability for accessing the bastion host, as you can rely on a fixed IP address instead of having to constantly update your access rules or DNS records.## EC2 Bastion Host Elastic IPresource "aws_eip" "ec2-bastion-host-eip" { vpc = true tags = { Name = "${var.project}-ec2-bastion-host-eip-${var.environment}" }}EC2 Instance & Elastic IP Association:Finally now we will create EC2 instance that we will be using as bastion host.resource "aws_instance" "ec2-bastion-host" { ami = "ami-0d76271a8a1525c1a" instance_type = "t2.micro" key_name = aws_key_pair.ec2-bastion-host-key-pair.key_name vpc_security_group_ids = [aws_security_group.ec2-bastion-sg.id] subnet_id = aws_subnet.vpc-public-subnet-2.id associate_public_ip_address = false user_data = file(var.bastion-bootstrap-script-path) root_block_device { volume_size = 8 delete_on_termination = true volume_type = "gp2" encrypted = true tags = { Name = "${var.project}-ec2-bastion-host-root-volume-${var.environment}" } } credit_specification { cpu_credits = "standard" } tags = { Name = "${var.project}-ec2-bastion-host-${var.environment}" } lifecycle { ignore_changes = [associate_public_ip_address,] }}The provided Terraform code snippet represents the definition of an AWS EC2 instance resource named ec2-bastion-host in the Terraform configuration. This resource is responsible for provisioning and managing an EC2 bastion host within the specified AWS environment.The EC2 instance is created with the Amazon Linux Amazon Machine Image (AMI) using the ami attribute and has an instance type of t2.micro. The SSH key pair for authentication is specified with key_name using the aws_key_pair.ec2-bastion-host-key-pair.key_name reference that we have created earlier. The EC2 instance is launched in the specified public subnet 2, to ensure it resides in the desired network segment. The option associate_public_ip_address is set to false, indicating that the EC2 instance won't have a public IP address associated with it.The configuration also defines the root block device attributes, including the volume size, deletion on termination, volume type, encryption, and associated tags. The credit_specification section sets the CPU credits to "standard" for the instance. Lastly, the lifecycle block is included to ignore changes in the associate_public_ip_address attribute during updates to prevent unnecessary modifications.To access our EC2 bastion host now we are going to associate Elastic IP that we have created earlier with our EC2 instance using following terraform resource block:## EC2 Bastion Host Elastic IP Associationresource "aws_eip_association" "ec2-bastion-host-eip-association" { instance_id = aws_instance.ec2-bastion-host.id allocation_id = aws_eip.ec2-bastion-host-eip.id}Now we are ready to apply the infrastructure, once the infrastructure is provisioned you can use you private key and the public IP of your EIP to access the EC2 bastion host as follows:ssh -i ec2-bastion-key-pair.pem ec2-user@Conclusion:In conclusion, an EC2 Bastion Host is an important component of a secure AWS architecture as it provides a secure and controlled way to access instances within a private subnet. Provisioning the EC2 Bastion Host using Terraform allows for easy management of infrastructure as code. In this article, we went through the process of provisioning an EC2 Bastion Host using Terraform and also highlighted the importance of using best practices, such as limiting incoming traffic and using an Elastic IP address. By following these best practices, we can ensure a secure and manageable infrastructure that is easy to maintain and scale as required. We hope this article has provided valuable insights into the EC2 Bastion Host and how to provision it using Terraform. With EC2 Bastion Host setup in our next article RDS: Deploying Scalable and Resilient Relational Databases Using Terraform IaC you can check how to deploy a relational database using AWS RDS.Recommended Readings:Final NoteIf you enjoyed this article and found it useful, be sure to follow me on Medium and GitHub for more content like this. On Medium, you can find more articles on Cloud Computing, DevOps, Machine Learning and other related topics. On GitHub, you can find my open-source projects and code samples. By following me on these platforms, you can stay up-to-date with my latest work and learn more about best practices for managing infrastructure with Terraform and other cloud tools. Thanks for reading! You cant perform that action at this time. In this example we'll generate a ssh key pair and use terraform to create the following resources. The goal is to be able to ssh to a bastion host and run a terraform provisioner to the private instance.NetworkVPCPublic SubnetPrivate SubnetInternet GatewayElastic IPNat GatewayRoute tablesRoute table associationsSecurity groups (ingress ssh and egress all)Ec2keypairBastion host (public Subnet)Private Instance (Private Subnet)Terraform cli installedAWS account with permissions to create the above resourcesOpen TerminalPaste the text belowssh-keygen -m PEM -f terraform_aws_bastion_ssh -N ''This will create the new ssh key In the same Terminal paste the text below subsituting your aws access key and secret access key.export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLEexport AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEYexport AWS_DEFAULT_REGION=ca-central-1 Typically we don't commit the .tfvars file to version control. For ease:Run the following command in the same Terminalcp terraform.tfvars.example terraform.tfvars Run the following commands in the same Terminalterraform initterraform validateterraform planReview the planned changes and run the following command You will see that after the bastion host has been provisioned, the private instance will then be provisioned. The terraform provisioner will connect to the private instance via the bastion host and run the inline scripts to setup the LAMP stack. See the code snippet below. provisioner "remote-exec" { inline = ["sudo yum update -y", "sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2", "cat /etc/system-release", "sudo yum install -y httpd mariadb-server", "sudo systemctl start httpd", "sudo systemctl enable httpd", "sudo systemctl is-enabled httpd"] } connection { host = self.private_ip type = "ssh" user = "ec2-user" private_key = file(var.ssh_private_key_path) bastion_host = aws_instance.my_bastion_instance.public_ip bastion_host_key = file(var.ssh_public_key_path) }Run the following commands in the same Terminal Are you a beginner and planning to learn Terraform? Do you want to know all the basic fundamental concepts of Terraform and how it works before you go diving deep, then this terraform tutorial blog post is for you!In this blog post, I am going to cover a brief introduction of Infrastructure as Code (IaC), Terraform, its lifecycle, and all the core concepts that every beginner should know. I have tried to cover all the topics in this beginners guide that will give you a quick start for using Terraform. What Is Infrastructure as Code (IaC)?Infrastructure as Code (IaC) is a widespread terminology amongDevOps professionals and a key DevOps practice in the industry. It is the process of managing and provisioning the complete IT infrastructure (comprises both physical and virtual machines) using machine-readable definition files. It helps in automating the complete data center by using programming scripts.Popular IaC Tools:1. TerraformAn open-source declarative tool that offers pre-written modules to build and manage an infrastructure.2. Chef: A configuration management tool that uses cookbooks and recipes to deploy the desired environment. Best used for Deploying and configuring applications using a pull-based approach.3. Puppet: Popular tool for configuration management that follows a Client-Server Model. Puppet needs agents to be deployed on the target machines before the puppet can start managing them.4. Ansible: Ansible is used for building infrastructure as well as deploying and configuring applications on top of them. Best used for Ad hoc analysis.5. Packer: Unique tool that generates VM images (not running VMs) based on steps you provide. Best used for Baking compute images.6. Vagrant: Builds VMs using a workflow. Best used for Creating pre-configured developer VMs within VirtualBox.Read our blog to know why Terraform is preferred over other IaC tools Terraform vs Ansible Kickstart Your Terraform Journey: Free Terraform MasterClass for Beginners!Enroll For Free Terraform is one of the most popular Infrastructure-as-code (IaC) tool, used by DevOps teams to automate infrastructure tasks. It is used to automate the provisioning of your cloud resources. Terraform is an open-source, cloud-agnostic provisioning tool developed by HashiCorp and written in GO language.Benefits of using Terraform:Does orchestration, not just configuration managementSupports multiple providers such as AWS, Azure, Oracle, GCP, and many moreProvide immutable infrastructure where configuration changes smoothlyUses easy to understand language, HCL (HashiCorp configuration language)Easily portable to any other providerCheck out our blog for everything you need to know about Terraform Certification Terraform CertificationTerraform LifecycleTerraform lifecycle consists of init,plan,apply, anddestroy.1. Terraform init initializes the (local) Terraform environment. Usually executed only once per session.2. Terraform plan compares the Terraform state with the as-is state in the cloud, build and display anexecution plan. This does not change the deployment (read-only).3. Terraform apply executes the plan. This potentially changes the deployment.4. Terraform destroy deletes all resources that are governed by this specific terraform environment.Terraform Core Concepts1. Variables: Terraform has input and output variables, it is a key-value pair. Input variables are used as parameters to input values at run time to customize our deployments. Output variables are return values of a terraform module that can be used by other configurations.Read our blog on Terraform Variables2. Provider: Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, OCI, and others. A provider is a plugin that interacts with the various APIs required to create, update, and delete various resources.Read our blog to know more about Terraform Providers3. Module: Any set of Terraform configuration files in a folder is a module. Every Terraform configuration has at least one module, known as itsroot module.4. State: Terraform records information about what infrastructure is created in a Terraform state file. With the state file, Terraform is able to find the resources it created previously, supposed to manage and update them accordingly.5. Resources: Cloud Providers provides various services in their offerings, they are referenced as Resources in Terraform. Terraform resources can be anything from compute instances, virtual networks to higher-level components such as DNS records. Each resource has its own attributes to define that resource.6. Data Source: Data source performs a read-only operation. It allows data to be fetched or computed from resources/entities that are not defined or managed by Terraform or the current Terraform configuration.7. Plan: It is one of the stages in the Terraform lifecycle where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.8. Apply: It is one of the stages in the Terraform lifecycle where it applies the changes real/current state of the infrastructure in order to achieve the desired state.Check Out:Our previous blog post on Terraform Cheat Sheet.Terraform InstallationBefore you start working, make sure you have Terraform installed on your machine, it can be installed on any OS, say Windows, macOS, Linux, or others. Terraform installation is an easy process and can be done in a few minutes.Read our blog to know how to install Terraform in Linux, Mac, WindowsWe cover the step-by-step Terraform installation in all these ways in our Terraform training. Check out our blog for all the Hands-on Labs that we cover in our training HashiCorp Certified Terraform Associate-Step By Step Activity Guides.Terraform ProvidersA provider is responsible for understanding API interactions and exposing resources. It is an executable plug-in that contains code necessary to interact with the API of the service. Terraform configurations must declare which providers they require so that Terraform can install and use them.Terraform has over a hundred providers for different technologies, and each provider then gives terraform user access to its resources. So through AWS provider, for example, you have access to hundreds of AWS resources like EC2 instances, the AWS users, etc.Read More:About Terraform Workflow.Terraform Configuration FilesConfiguration files are a set of files used to describe infrastructure in Terraform and have the file extensions .tf and .tf.json. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.A Terraform configuration is made up of one or more files in a directory, provider binaries, plan files, and state files once Terraform has run the configuration.1. Configuration file (*.tf files): Here we declare the provider and resources to be deployed along with the type of resource and all resources specific settings2. Variable declaration file (variables.tf or variables.tf.json): Here we declare the input variables required to provision resources3. Variable definition files (terraform.tfvars): Here we assign values to the input variables4. State file (terraform.tfstate): a state file is created once after Terraform is run. It stores state about our managed infrastructure.Also Read:Our blog post on Terraform Create VM.Getting started using TerraformTo get started building infrastructure resources using Terraform, there are few things that you should take care of. The general steps to deploy a resource(s) in the cloud are:Set up a Cloud Account on any cloud provider (AWS, Azure, OCI)Install TerraformAdd a provider AWS, Azure, OCI, GCP, or othersWrite configuration filesInitialize Terraform ProvidersPLAN (DRY RUN) using terraform planAPPLY (Create a Resource) using terraform applyDESTROY (Delete a Resource) using terraform destroyAlso Check:Our blog post on Terraform Interview Question.Import Existing InfrastructureTerraform is one of the great IaC tools with which, you can deploy all your infrastructures resources. In addition to that, you can manage infrastructures from different cloud providers, such as AWS, Google Cloud, etc. But what if you have already created your infrastructure manually?Terraform has a really nice feature for importing existing resources, which makes the migration of existing infrastructure into Terraform a lot easier.Currently, Terraform can only import resources into the state. It does not generate a configuration for them. Because of this, prior to running terraform import it is necessary to write manually a resource configuration block for the resource, to which the imported object will be mapped. For example:resource "aws_instance" "import_example" { # ...instance configuration... }Now terraform import can be run to attach an existing instance to this resource configuration:$ terraform import aws_instance.import_example i-03efafa258104165fThis command locates the AWS instance with ID i-03efafa258104165f (which has been created outside Terraform) and attaches it to the name aws_instance.import_examplein the Terraform state.Check Out:Our blog post on Terraform Tips and Tricks.ConclusionI hope the above gives you an idea about how you can get started with Terraform. If you are interested in learning more, then I would suggest checking out our training Cloud Infrastructure Automation Certification: Terraform Associate Training where we have covered all these topics in detail and much more along with Hands-on labs on each topic.Frequently Asked Questions No, prior programming or infrastructure experience is not necessary to follow the guide. It is designed to cater to beginners and assumes no prior knowledge of Terraform. The guide provides step-by-step explanations and examples to help newcomers understand and apply the concepts effectively. The guide may mention a few prerequisites, such as having a basic understanding of cloud computing concepts and having an account with a cloud provider (if you plan to provision resources in the cloud). Additionally, it may recommend installing Terraform and a text editor suitable for writing code. Yes, the Terraform Beginner's Guide typically includes hands-on examples and exercises throughout the content. These examples help solidify the concepts and allow readers to practice writing Terraform configurations, executing commands, and managing infrastructure resources. Infrastructure as Code tools typically handle updates and changes by comparing the desired state defined in the code with the current state of the infrastructure. When changes are made to the code, the tools generate an execution plan that outlines the modifications required to achieve the desired state. This plan can be reviewed and then applied to update or modify the infrastructure accordingly. Yes, Infrastructure as Code can be used for existing infrastructure. By defining the existing infrastructure in code, you can capture its current state and make modifications to it using code-based configuration files. This approach allows you to manage existing infrastructure in a consistent and automated manner. Related/References Join FREE Class Master Terraform & DevOps to get High-Paying Jobs! Join our EXCLUSIVE Free class! Get your hands dirty with lots of projects and labs based on Terraform and DevOps in our Program.Click on the below image to Register for Our FREE Class Now!Terraform is an infrastructure as code tool that lets you build, change, and version infrastructure safely and efficiently. This includes low-level components like compute instances, storage, and networking; and high-level components like DNS entries and SaaS features.Learn more In our previous post, we explored how to enhance storage resiliency by attaching EFS volumes to multiple EC2 instances. Now, let's delve into AWS networking concepts, starting with the pivotal role of Bastion hosts. We'll employ Terraform to create modular components, ensuring a seamless and reproducible setup. A bastion host is a server whose purpose is to provide access to a private network from an external network, such as the Internet. Because of its exposure to potential attack, a bastion host must minimize the chances of penetration. For example, you can use a bastion host to mitigate the risk of allowing SSH connections from an external network to the Linux instances launched in a private subnet of your Amazon Virtual Private Cloud (VPC). Architecture Overview: Before diving into implementation, let's understand the architecture we'll be building: Step 1: Creating the VPC and Network Components Create VPC with IGW, 1 Public and 1 private subnet with route table associations. Please refer to my github repo in resources section below. Step 2: Deploying a Bastion Host in the Public Subnet and a Private Host in the Private Subnet Deploy Linux EC2 instances one in each subnet. ## Create EC2 Server Instances## module "vpc_a_bastion_host" { source = "./modules/web" instance_type = var.instance_type instance_key = var.instance_key subnet_id = module.vpc_a.public_subnets[0] vpc_id = module.vpc_a.vpc_id ec2_name = "Bastion Host A" sg_ingress_ports = var.sg_ingress_public common_tags = local.common_tags naming_prefix = local.naming_prefix} module "vpc_a_private_host" { source = "./modules/web" instance_type = var.instance_type instance_key = var.instance_key subnet_id = module.vpc_a.private_subnets[0] vpc_id = module.vpc_a.vpc_id ec2_name = "Private Host A" sg_ingress_ports = var.sg_ingress_private common_tags = local.common_tags naming_prefix = local.naming_prefix} Step 3: Implementing Access Restriction using Security Groups Amend private subnet security group to allow traffic only from public subnet. ## Amend Private Host SG to allow traffic from Bastion Host SG##resource "aws_security_group_rule" "public_in_ssh" { type = "ingress" from_port = 22 to_port = 22 protocol = "tcp" security_group_id = module.vpc_a_private_host.security_group_id source_security_group_id = module.vpc_a_bastion_host.security_group_id} Steps to Run Terraform Follow these steps to execute the Terraform configuration: terraform initterraform plan terraform apply -auto-approve Upon successful completion, Terraform will provide relevant outputs. Apply complete! Resources: 12 added, 0 changed, 0 destroyed. Testing the outcome Bastion Host with Public IP in Public Subnet: Private Host without Public IP in Private Subnet: Private Host Security group with only inbound from Bastion Host Security Group: Connecting to Private Host from Bastion Host. You will need to create key pair with correct permissions on bastion host before connecting to Private Host chmod 0400 WorkshopKeyPair.pemssh ec2-user@10.1.2.71 -i WorkshopKeyPair.pem Cleanup: Remember to stop AWS components to avoid large bills. terraform destroy -auto-approve With the Bastion host setup accomplished, our next module will delve deeper into AWS networking. We'll explore setting up peer-to-peer VPC connections to further enhance our network architecture. Resources: Github Link: Host Concept: As the software development landscape evolves in 2025, organisations are increasingly focused on optimising the way they build, deploy, and manage applications. Please enable Javascript to use this application In this article, we will discuss what a Bastion architecture is, the challenges it solves, and we will have fun implementing it on AWS using Terraform.We will also see how to have a backup in case of loss of the Bastion host, using AWS Session Manager.This article is made to be light and fun. It will be enough to introduce you to secure cloud architectures.This article contains an intermediate-level lab, we assume that you have knowledge of AWS (VPC, EC2, Subnets..) and Terraform, and are comfortable with SSH.Pre-requisitesBefore we dive into the main subject, lets make sure you have the necessary tools for this lab.AWS account and Access KeysAWS CLITerraformSSH CLI and SSH keypairChallengesIn a cloud environment, securing resources is the most crucial part of building architectures. Imagine working in a multi-billion company and having an EC2 instance hacked and secret data used. The companys stocks will instantly crash, starting a crisis and probably putting an end to this company.Now that we have a real impact of what an insecure cloud architecture would make on a company, lets take it into a smaller context.Say you have a personal AWS VPC, with 3 EC2 instances. The instances are in a public subnet, you access them using SSH (via public internet) to make development and maintenance operations.One day, you started to observe intrusion attempts made by strange sources. So you decide to put these instances in a private subnet thats not accessible from the outside world. Now your EC2 instances are perfectly secure, but How would you access them? You lost your access.Here comes the role of a Bastion host.A Bastion host is a specially configured server designed to act as a gateway between an untrusted network, in our case the internet, and a private trusted network, in our case AWS private subnets.Key featuresBastion's key features are:Single Entry Point: channels all external traffic through its secure gateway.Hardened Security: minimal services installed to reduce vulnerabilities, including firewalls, intrusion detection softwares, and authentication systems.Access Control: It verifies and limits access to authorized users.Audit and Monitoring: Logs and monitors incoming and outgoing traffic.ArchitectureLets discuss the architecture component by component1 VPC main-vpc1 private subnet with the CIDR 192.168.1.15 private-instances-subnet3 EC2 instances private{001,002,003} in the subnet private-instances-subnet1 public subnet CIDR 192.168.1.16 bastions-subnet1 EC2 instance bastion001 in the subnet bastions-subnet1 Internet gateway main-igw1 NAT gateway main-nat-gateway1 Local desktop yours :)ImplementationIve prepared for you all the necessary Terraform configuration and other resources, so we can focus on understanding them.The code is located on GitHub aws-bastion-architecture.Clone the repositorygit clone git@github.com:rafikbahri/aws-bastion-architecture.gitThe project structure is as follow# --- Project Structure ---bastion.tf # bastion configuration (subnet, instances, sg..)main.tf # defines the provider (aws)outputs.tf # terraform outputs (bastions public ips, private instances ips..)private.tf # private instances configuration (subnet, instances, sg..)ssh-config.tf # generate ssh config file located in .ssh/configssm.tf # aws session manager configuration (sg..)variables.tf # global variablesvpc.tf # vpc configuration (subnets, sg, vpc, igw...)[.config] cloudinit_user_data.yaml # contains you public ssh key[modules] # custom terraform modules [aws-node] # encapsulates an ec2 instnace/node configuration README.md main.tf outputs.tf variables.tf versions.tf [aws-private-subnet] # encapsulates a private subnet configuration README.md main.tf outputs.tf variables.tf versions.tf [aws-public-subnet] # encapsulates a public subnet configuration README.md main.tf outputs.tf variables.tf versions.tf [aws-sg] # encapsulates a security group configuration README.md main.tf outputs.tf variables.tf versions.tf [aws-vpc] # encapsulates a vpc configuration README.md main.tf outputs.tf variables.tf versions.tfWell explain the major important parts of the project, enjoy reading the code on your side and try to understand every little bit of it.Lets dive deep into the bastion.tf configuration fileIn the first part, we define the bastions-subnet that uses a module to create an AWS public subnet inside the VPC we create using vpc.tf.Then we define a group of instances, in our case var.bastion_servers_count is equal to 1. Check the variables.tf file for all the details on the values of the variables.We can scale the server count to 3, providing the private_ips which is a list of IPs inside the subnet, which has a CIDR of 192.168.15.0/24.The user_data_file is a crucial part of this configuration since it has our public SSH key. This file has a configuration Shell script that is run during the startup of the EC2 instances.Youll need to override the value of SSH_PUBLIC_KEY inside the .config/cloudinit_user_data.yaml file with your own SSH public key.The bastion host has 2 security groups: module.sg-admin-bastions.sg_id, module.sg-admin.sg_id.module "bastions-subnet" { source = "./modules/aws-public-subnet" name = "bastion-subnet" vpc_id = module.main-vpc.vpc_id availability_zone = "eu-west-3a" cidr_block = var.bastions_subnet_cidr map_public_ip_on_launch = true public_internet_route_table_id = module.main-vpc.public_internet_route_table_id has_internet_access = true tags = { group = "bastions" }}module "bastions" { source = "./modules/aws-node" server_count = var.bastion_servers_count server_prefix = "bastion" ami_id = "ami-0546127e0cf2c6498" instance_type = "t2.micro" vpc_id = module.main-vpc.vpc_id subnet_id = module.bastions-subnet.subnet_id private_ips = [["192.168.15.11"], ["192.168.15.12"], ["192.168.15.13"]] create_key = false security_groups = [module.sg-admin-bastions.sg_id, module.sg-admin.sg_id] user_data_file = ".config/cloudinit_user_data.yaml" tags = { purpose = "bastion" description = "Serves for SSH access" component = "infra" }}Lets understand the security groups.module "sg-admin-bastions" { source = "./modules/aws-sg" name = "sg_admin_bastions" description = "Admin security group" vpc_id = module.main-vpc.vpc_id ingress_rules = [{ description = "SSH only from admin IP addresses." from_port = 22 to_port = 22 protocol = "tcp" cidr_blocks = ["88.178.215.32/32" # My public IP address] # Required attribues: ipv6_cidr_blocks = [] prefix_list_ids = [] security_groups = [] self = false }, { description = "Ping inside VPC." from_port = 0 to_port = 0 protocol = "icmp" cidr_blocks = ["192.168.0.0/16"] ipv6_cidr_blocks = [] prefix_list_ids = [] security_groups = [] self = false }] egress_rules = [{ description = "Allow all outbound traffic." from_port = 0 to_port = 0 protocol = "-1" cidr_blocks = ["0.0.0.0/0"] ipv6_cidr_blocks = [] prefix_list_ids = [] security_groups = [] self = false }]}module "sg-admin" { source = "./modules/aws-sg" name = "sg_admin" description = "Admin security group incoming only from bastions" vpc_id = module.main-vpc.vpc_id ingress_rules = [{ description = "SSH using EC2 instance connect (from AWS console)." from_port = 22 to_port = 22 protocol = "tcp" cidr_blocks = ["35.180.112.80/29" # EC2 instance connect service IPs in my region] ipv6_cidr_blocks = [] prefix_list_ids = [] security_groups = [module.sg-admin-bastions.sg_id] self = false }, { description = "Ping inside VPC." from_port = -1 to_port = -1 protocol = "icmp" cidr_blocks = ["192.168.0.0/16"] ipv6_cidr_blocks = [] prefix_list_ids = [] security_groups = [] self = false }] egress_rules = [{ description = "Allow all outbound traffic." from_port = 0 to_port = 0 protocol = "-1" cidr_blocks = ["0.0.0.0/0"] ipv6_cidr_blocks = [] prefix_list_ids = [] security_groups = [] self = false }]}sg-admin-bastions allows SSH access to the bastion host only from my public IP address. So youll need to change that value by your own public IP address.sg-admin allows EC2 Instance Connect as a backup method in case we lose our SSH keys. (Totally optional but its a nice facility in this architecture)Now lets understand the private instances configurationmodule "private-instances-subnet" { source = "./modules/aws-private-subnet" name = "private-instances-subnet" vpc_id = module.main-vpc.vpc_id availability_zone = "eu-west-3a" cidr_block = var.private_instances_subnet_cidr public_subnet_id = module.bastions-subnet.subnet_id has_internet_access = true tags = { kind = "private" }}module "private-instances" { source = "./modules/aws-node" server_count = var.private_instances_count server_prefix = "private" ami_id = "ami-0546127e0cf2c6498" instance_type = "t2.micro" vpc_id = module.main-vpc.vpc_id subnet_id = module.private-instances-subnet.subnet_id private_ips = [["192.168.16.11"], ["192.168.16.12"], ["192.168.16.13"]] create_key = false security_groups = [module.sg-admin.sg_id] user_data_file = ".config/cloudinit_user_data.yaml" tags = { kind = "private" }}This is a typical configuration, we define a subnet and a group of nodes/instances. We initialize the EC2 instances with the .config/cloudinit_user_data.yaml and we configure security groups to use module.sg-admin.sg_id so we only accept SSH from the Bastion host.Lastly, before we attack the practical part, lets talk about ssm.tf.This file configures AWS Session Manager for all of our EC2 instances.Session Manager is a service that gives us access to EC2 instances without the need for SSH.Its a great backup method if we lose our SSH keys or Bastion host.Initialize Terraform resourcesterraform initIts a good practice to generate a terraform plan and read it before applying the configuration. Lets go ahead and runterraform planOnce we are sure of our configuration, we can go ahead and apply itterraform applyThe `terraform apply` command runs the plan command and asks for explicit confirmation by typing `yes`Test the architectureOn your AWS Console, open EC2 > Instances (Running)EC2 > Instances (Running)We can see that only the bastion host bastion001 has a Public IPv4.Test your SSH access to the Bastion hostssh -F .ssh/config ec2-user@bastion001Test your SSH access to a private instancessh -F .ssh/config ec2-user@private001You should have a similar output toSSH access testsGo ahead and test your access using AWS Session Manager as wellTo do so, choose an EC2 instance, then Connect > Session ManagerConnectConnect Session ManagerSession Manager testConclusionWe understood what a Bastion host is, its challenges, and key features.We also saw how to implement a typical Bastion host architecture on AWS and have a backup thanks to AWS Session Manager, all this using the famous Terraform.I hope you enjoyed this article, feel free to give me your feedback and whether you want me to make similar articles on other Cloud architectures on AWS using Terraform.Thanks a lot, see you in other tech articles!Security is extremely important. Ensuring security is a critical aspect of our infrastructure, which is why we deploy certain resources in private subnets, that are not accessible from the internet. However, there are scenarios where we still require access to these resources over the Internet.For instance, when creating an Amazon EC2 instance or an Amazon RDS DB instance in a private subnet (without internet connectivity and public IPs), accessing them becomes challenging. In such cases, there are several solutions available to connect these private resources within the Amazon Web Services (AWS) environment.The preferred approach is establishing connectivity through a Virtual Private Network (VPN) or AWS Direct Connect, which offers secure connections. However, if you do not have VPN or Direct Connect configured, an alternative option is to utilize a Bastion Host, That can help add a layer of security to your AWS instances is the Bastion host.A bastion host is a server whose purpose is to provide access to a private network from an external network, such asTerraform is an infrastructure as code tool that lets you build, change, and version infrastructure safely and efficiently. This includes low-level components like compute instances, storage, and networking; and high-level components like DNS entries and SaaS features.Learn more Share copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Follow the documentationto update the SSH configuration file to allow SSH connections through Session Manager.It allows running a proxy command that starts a Session Manager session and transfer all data through opened connection. Generate local SSH private and public keys. For example, you can use following command:ssh-keygen -t rsa -f my_keyIt will generate private and public SSH key pair which are going to be used to connect to bastion host.It is recommended to provide password to protect access to keys and store keys in secure location. In order to connect to deployed bastion host you will need to obtain EC2 instance id. There are multiple ways you can dothat. For example you can get it using AWS console by navigatingto EC2 dashboard or through AWS CLI using:aws ec2 describe-instancesTo further filter results you can use following command:aws ec2 describe-instances --filters 'Name=tag:Name,Values=$BASTION_HOST_TAG' --output text --query 'Reservations[*].Instances[*].InstanceId' --output textReplace $BASTION_HOST_TAG with tag used to mark bastion host. Default value is sandbox-dev-bastion-host.Copy obtained EC2 instance id for later use. In order to connect to the bastion host we first have to send SSH key to the host using EC2 Instance Connect.Use following command replacing $INSTANCE_ID with EC2 instance id obtained in previous step and $PUBLIC_KEY_FILEwith path to your public key file (for example: my_key.pub).Be sure to use public key and NOT private key.aws ec2-instance-connect send-ssh-public-key --instance-id $INSTANCE_ID --instance-os-user ec2-user --ssh-public-key file://$PUBLIC_KEY_FILEYou should receive message indicated successful upload of key. You have just uploaded temporary SSH keyto EC2 instance metadata where itsgoing to remain for 60 seconds. After 60 seconds SSH key gets removed automatically, and you wont be able to use it toconnect to the instance. You will see Permission denied error if you try. If this happens you can resend the key usingthe same command.This means that you have 60 seconds to initialize SSH connection after you upload keys. Follow next step to do so. In this last step you will connect to your bastion host using SSH. Use following command replacing$PRIVATE_KEY_FILE with path to your private key (for example: my_key) and $INSTANCE_ID with EC2 instance id obtainedinprevious steps.ssh -i $PRIVATE_KEY_FILE ec2-user@$INSTANCE_IDConfirm connection by typing yes. It will open SSH connection using previously configured Session Manager.Youre in! It is possible to use different options to open connection to bastion host. For example you can use -D 8888 option toopen SSH connection with a local dynamic application-level port forwarding through 8888 port.See this link fordetailed explanation.ssh -i $PRIVATE_KEY_FILE -D 8888 ec2-user@$INSTANCE_IDThis is kind of connection opens SOCKS proxy you can use for example to forward traffic from your local browser throughbastion host. Refer to man pages of ssh command to see all options. In order to remove all deployed resources run the following command from the root directory of this pattern:terraform destroy -var-file="dev.tfvars"and confirm removal of resources. Instead of manually executing steps described above, you can use provided connect.sh script. It willgenerate set of SSH keys, push them to EC2 instance and initiate connection with the host. Execute the script and passtag and key name as arguments. Follow the prompt to connect to your bastion host. Example:./connect.sh sandbox-dev-bastion-host my_key Sometimes you might experience TargetNotConnected error when trying to connect to the bastion host.Solution: Logging and monitoring are important parts of maintaining systems both from the operational and security perspectives.There are multiple ways in which you can monitor connections and activity in your bastion host. Below you can find someof the resources from AWS documentation that are related to this topic: You cant perform that action at this time.

Aws bastion host terraform. Bastion host aws.