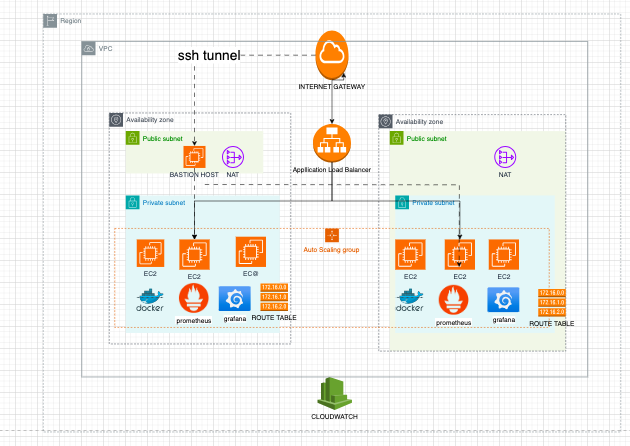
**Infrastructure Building And Java Application Deployment.**

Architecture -



Architecture Discussion -  
 The architecture chosen is a highly available, encrypted and automated one for infrastructure building and deployment.   
  
We start with building VPC with cidr range, subnets (public and private) with their respective cidr blocks in two different availability zones to make the application highly available. Then, Internet gateway is built for the internet access for vpc followed by creating two elastic Ips for a nat gateway which are deployed in public subnets for facilitating instances in the private subnet to access internet safely. Route tables are built and attached to the respective subnets for the proper traffic flow in and across the subnets.  
  
We then create aws security rule for security groups to avoid cycle conflicts when mentioned in the security group as ingress and egress. We create a security group for bastion host as it needs separate permissions/rules to access private instance. Moving further, key bair is generated and stored in the local path where the terraform has run. Bastion host instance is created along with a launch template of ec2 instance for private subnet which are created through auto scaling groups.  
  
Load balancer is created to route the traffic and then auto scaling is assigned to the load balancer. You have to set metrics in the auto scaling groups for you scale in and scale out feature. Its default or not set in the current code as it varies. Best practise is to set the metric for cpu+memory to hit 75%.

User perspective-   
Traffic comes from the internet gateway to the load balancer in public subnet and then goes to the instances in the private subnet, where it gets the response and goes back in the same route.  
  
Developer’s Perspective -   
Traffic goes from internet gateway and then to bastion host (as you cannot reach private instance through load balancer while deploying an application), from the bastion host it reached the private instances where the code is deployed and necessary updates or downloads are done by obtaining the internet flow through nat gateways in public subnet. In our case, the docker is downloaded in private subnets using terraform userdata is achieved through internet flow for private instances through Nat Gateway. Port for prometheus and grafana are kept open in bastion host security groups and create a secure ssh shell between bastion host and private instances by modifying the security group as needed(explained in later stages of this documentation). We do not open the ports for prometheus and grafana in load balancer because they are tools used to monitor and visualize application performance, which is not for users.

Resource Table -

| Resources | Comments |
| --- | --- |
| VPC | 1 vpc is |
| Subnets (Public & Private) | 4 subnets |
| Application load balancer | 1 application load balancer |
| Internet gateway | 1 internet gateway |
| Route Table | 4 route tables for each subnet |
| Nat gateway | 2 nat gateway for each public subnet |
| Auto scaling groups | 1 auto scaling group with a launch template attached |
| Availability zones | 2 availability zones have been used to deploy resources. |

**Steps To Achieve The Output -**

What is Terraform?

* Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It enables users to define and provision infrastructure resources using a declarative configuration language. Essentially, Terraform allows you to manage your infrastructure in a programmable and version-controlled manner, similar to how you manage your application code.

### **Key Concepts:**

1. Infrastructure as Code (IaC):
   * Terraform embraces the concept of Infrastructure as Code, which means defining and managing infrastructure resources (such as virtual machines, networks, storage, etc.) using code.
   * With Terraform, you can describe your desired infrastructure state in configuration files, which can be version-controlled, reviewed, and shared like any other code.
2. Declarative Configuration Language:
   * Terraform uses a declarative language called HashiCorp Configuration Language (HCL) to define infrastructure resources and their configurations.
   * In HCL, you specify what infrastructure you want (e.g., AWS EC2 instances, VPCs, subnets) rather than specifying how to achieve it.
3. Resource Providers:
   * Terraform supports a wide range of cloud and infrastructure providers, including AWS, Azure, Google Cloud Platform (GCP), VMware, Kubernetes, and many others.
   * Each provider offers a set of resources and data sources that Terraform can manage.
4. Execution Plans:
   * Terraform generates an execution plan before making any changes to your infrastructure.
   * The execution plan provides a preview of what actions Terraform will take to create, modify, or delete resources to achieve the desired state.
5. State Management:
   * Terraform maintains a state file that records the current state of your infrastructure.
   * The state file is used to track resource metadata, dependencies, and mapping between your configuration and real-world infrastructure.
   * State management is critical for Terraform to understand the existing state of your infrastructure and to make changes safely and predictably.
6. Immutable Infrastructure:
   * Terraform promotes the principle of immutable infrastructure, where infrastructure components are treated as disposable and are replaced rather than modified.
   * When you need to make changes to your infrastructure, Terraform creates entirely new resources and replaces the existing ones, which helps ensure consistency and reliability.

### **Workflow:**

1. Define Configuration:
   * Write Terraform configuration files to define the desired infrastructure resources, providers, and configurations.
2. Initialize:
   * Run terraform init to initialize the Terraform working directory and download provider plugins.
3. Plan:
   * Generate an execution plan with terraform plan to preview the changes Terraform will make to your infrastructure.
4. Apply:
   * Apply the changes with terraform apply to create, modify, or delete resources based on the execution plan.
5. Manage State:
   * Terraform automatically manages the state file, but it's crucial to store it securely and manage it properly, especially in a team environment.

Terraform simplifies and automates infrastructure management, enabling teams to provision and manage complex infrastructure resources efficiently, consistently, and reliably. It's widely used in DevOps, cloud-native, and modern software development workflows.

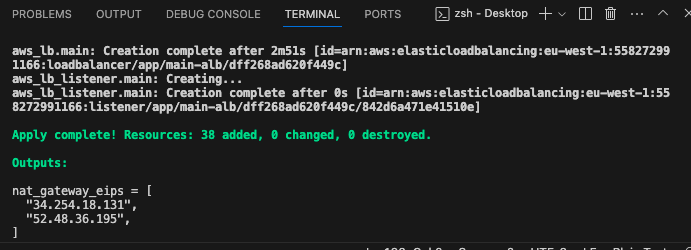
**Terraform**

Step 1 - Clone or fork the repository <https://github.com/sathvikbhupal1/AppDeployement> on your local machine or on the server where your terraform is installed.  
  
Step 2 - Run ‘terraform init’ in the folder you have cloned and that should initialize all the terraform plugins and providers based on the script.

Step 3 - Run “terraform validate” to check the syntax and connection.

Step 4 - Create a folder names secrets.tf vars and paste down your access key and secret access key in that.(for your reference)  
  
Step 4 - Run “terraform plan” which does the dry run and projects the changes that are going to happen. You will be asked to input your access key and secret access key.

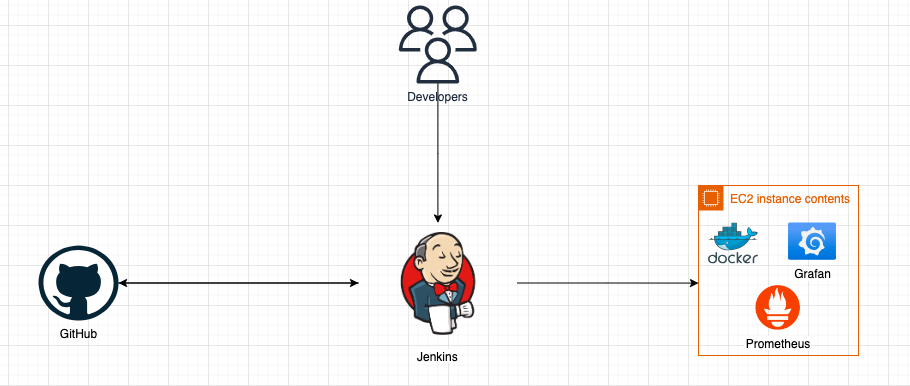
Step 5 - Run “terraform apply” and it should prompt for the access key and secret access key again, enter the necessary details. Yor infrastructure will be built with terraform on your aws account. A state file will be generated to track the resources you have built. A Key pair will be generated in the same folder which can be used to access the ec2 instances through bastion host.

  
  
Additional / Optional steps-  
  
Step 1 - Install jenkins on the server of your choice, and access it through ‘local host/ipaddr:8080’ through your browser.  
  
Step 2 - Follow the instructions on the jenkins UI to fetch the password, chage it and to install necessary plugins.  
  
Step 3 - You can create a new poject in jenkins and choose pipeline in the options.  
  
Step 4 - Copy paste the pipeline\_for\_terraform contents groovy file (without explanation in the end ) and do the necessary modifications on your local code in variable.tf and providers.tf accordingly.  
  
Step 5 - Run the pipeline and it should build the resources for you.  
  
1.You can even choose parameterised values in the options while creating the pipeline so that you can name the parameter and give the input values as apply and destroy. While on the script, change terraform apply to terraform $(parameter\_name). This will ask you to choose a parameter between apply and destroy on the time you want to build the pipeline.

2. You can change the state file storage to a remote backend on your aws S3 and lock it with dynamoDB by creating a backend.tf locally and writing the code for the backend using terraform docs for backend. Maintaining a remote backend is one of the best practices in terraform.

What is Jenkins?

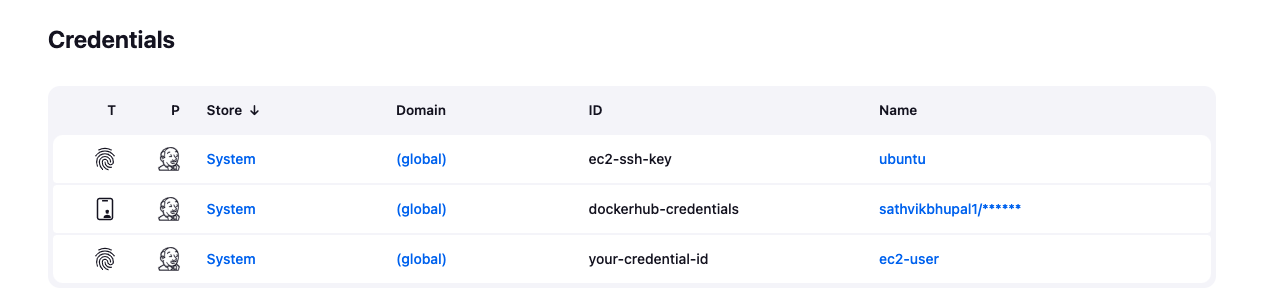
* Jenkins is an open-source automation server widely used for continuous integration (CI) and continuous delivery (CD) of software applications. It helps automate various stages of the software development lifecycle, including building, testing, and deploying applications. Jenkins facilitates the process of integrating code changes from multiple developers into a shared repository by regularly building and testing the application codebase to identify integration errors and conflicts early in the development cycle.

Additionally, it automates the deployment of applications to various environments, such as development, testing, staging, and production, ensuring that code changes are delivered to production quickly and reliably through automated deployment pipelines. Jenkins provides a vast ecosystem of plugins that extend its functionality to integrate with various tools and technologies, supporting a wide range of use cases, including version control systems, build tools, testing frameworks, and deployment platforms. It also supports distributed builds, allowing the distribution of build and test tasks across multiple machines or agents for parallel and concurrent execution, improving overall build performance and resource utilization. With extensive monitoring and reporting features, Jenkins offers visibility into the health and quality of software projects, enabling teams to identify issues and make data-driven decisions. Its robust security features, including authentication, authorization, and access control, ensure the confidentiality and integrity of CI/CD pipelines and resources, making Jenkins a valuable tool for enabling DevOps practices and accelerating the pace of software development and delivery.  
  
  
**Jenkins -**

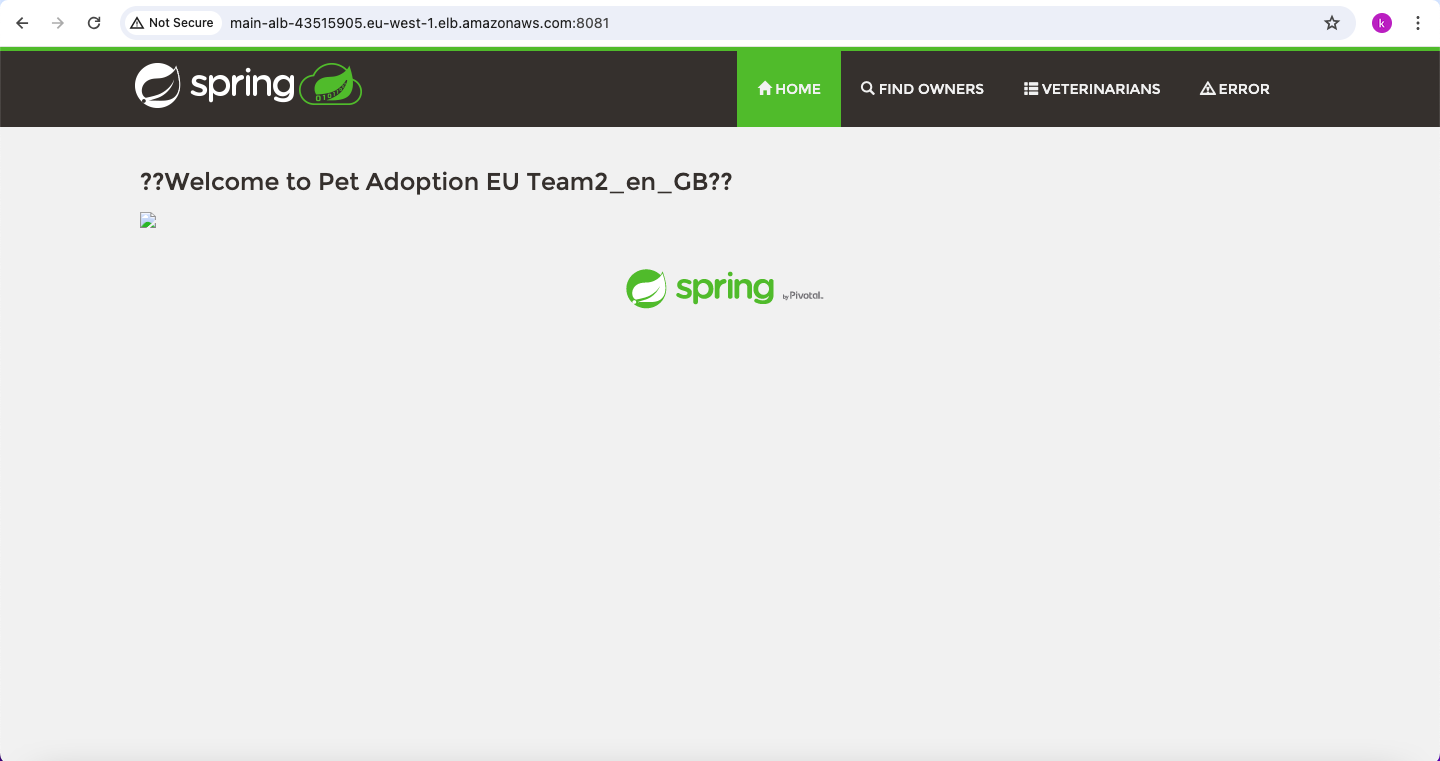
Step 1 - Install jenkins on the server of your choice, and access it through ‘local host/ipaddr:8080’ through your browser.

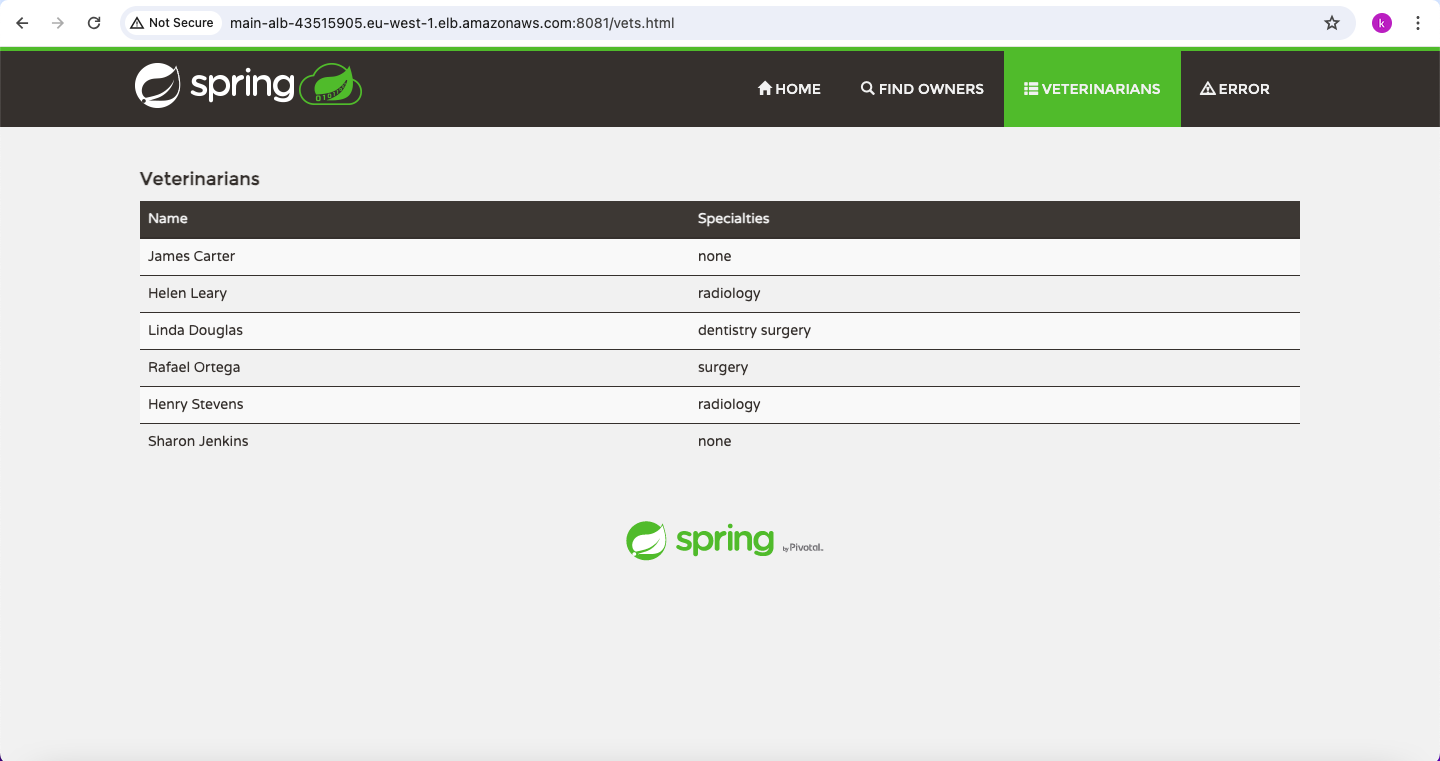
Step 2 - Follow the instructions on the jenkins UI to fetch the password, chage it and to install necessary plugins.

Step 3 - Go to manage jenkins -> credentials -> global credentials and add necessary credentials(environment variables) as shown in the picture below.

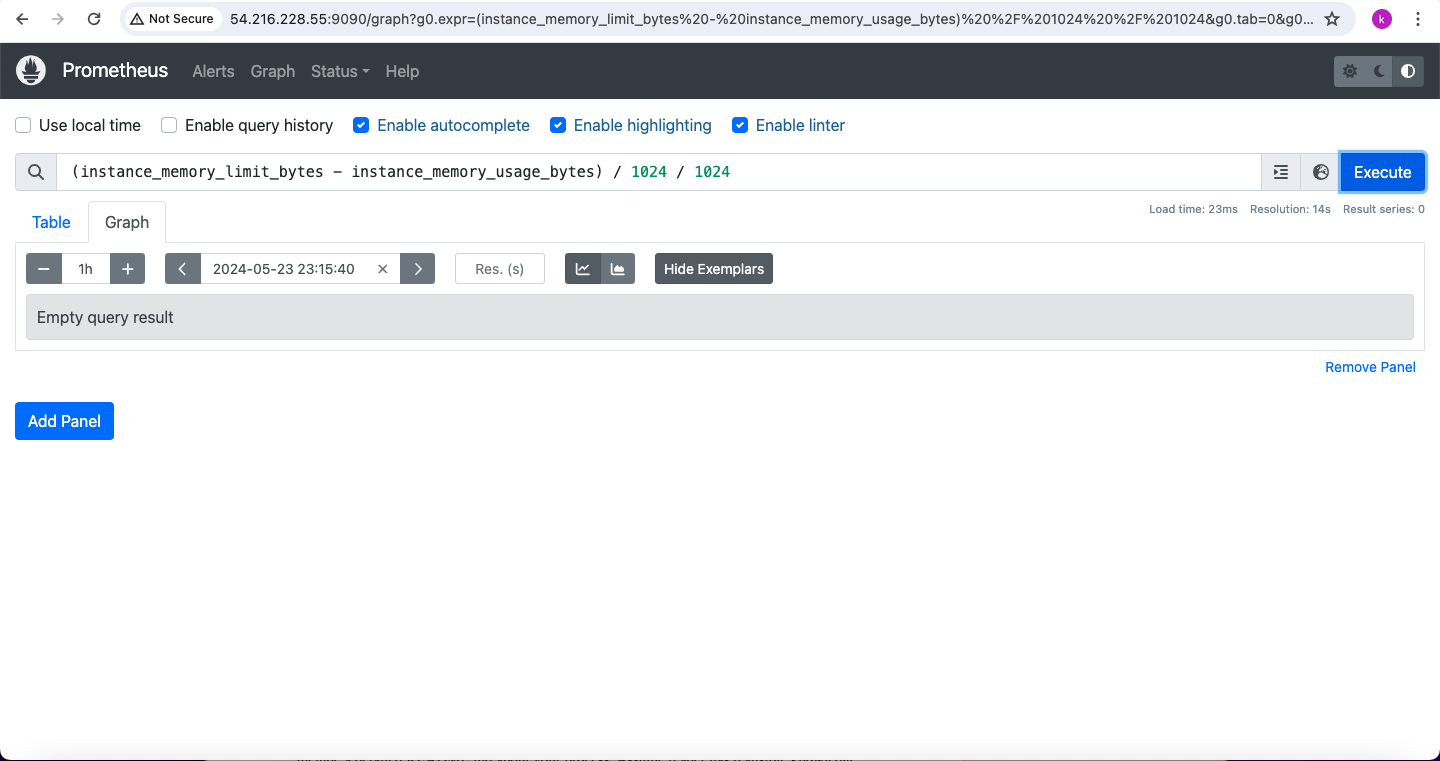


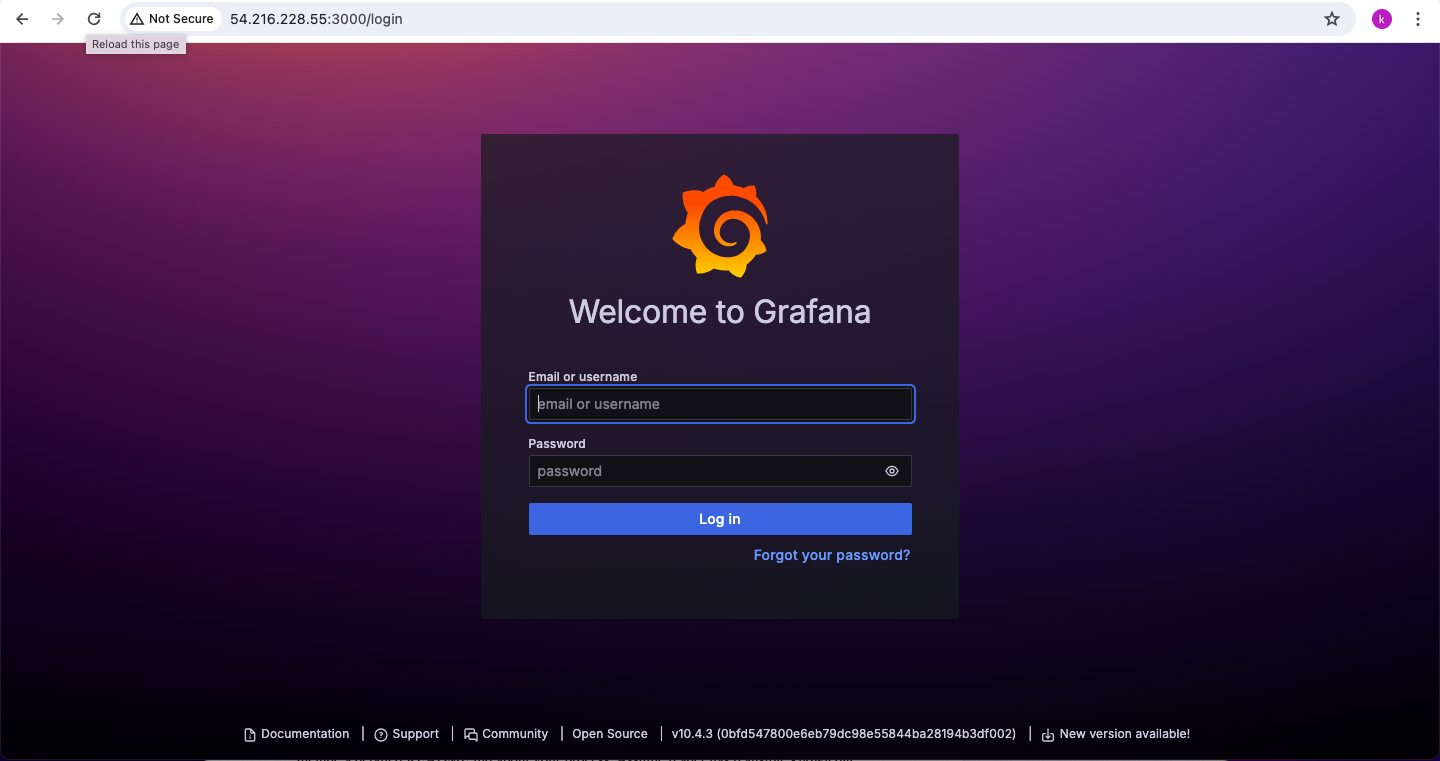
Step 4 - You can create a new poject in jenkins and choose pipeline in the options.

Step 5 - Copy paste the pipeline\_for\_appDeploy contents groovy file.  
  
Step 6 - Add necessary information for the pipeline such as your github link for the check out and bastion host ip and username, private ips of the instances created from terraform (taken from outputs value of your previous terraform apply) as we are hardcoding the values of the terraform to deploy the code in the container which will be running on the private instance you created through terraform.  
  
Step 7 - Build the pipeline and it should deploy the container on your instance successfully.  
  
Step 8 - Go to your application load balancer on your aws account, copy the dns from the resource and paste it on your browser. Add :8080 at last of your alb dns name and click enter. You should be able to view the website as seen in the underlying figures.  
  


Step 9 - You can also add /vets.html to your dns name to check the code further and it should display similar to the following figure.  
  


(Optional steps as prometheus and grafana should not be visible to end users but just development team)  
  
Step 9 - Additionally You can create a ssh tunnel between bastion host and private instances with modifying security group of private ec2 instance to allow access from bastion host port and ip (8081,ip address)(ingress) and egress to the bastion host port and ip address. You should also set egress of bastion host with private instances port and ip.

As per the code we have our application running on port 8081, prometheus running on port 9090 and grafana running on port 3000. On the public instance security group allow all traffic (ingress) on ports 9090, 8081 and 3000  
If you paste your ‘public ip:9090’ and ‘public ip:3000’ on your browser, you should see something similar to the underlying figure.  
  
  




Thus, achieving the desired output.   
  
**Pipeline Explaination -**

### **1. Environment Setup:**

* DOCKER\_IMAGE: Specifies the Docker image repository and tag.
* DOCKERHUB\_CREDENTIALS\_ID: Jenkins credentials ID for Docker Hub authentication.
* BASTION\_PUBLIC\_IP: Public IP address of the bastion host.
* BASTION\_USER: Username to access the bastion host.

### **2. Checkout:**

* This stage checks out the source code from the specified GitHub repository.

### **3. mvn clean:**

* Performs a Maven clean and package to build the Java application, skipping tests and checkstyle.

### **4. Docker Build and Push:**

* Builds the Docker image from the Dockerfile in the checked-out repository.
* Uses Docker Hub credentials to log in and push the Docker image to the specified repository.

### **5. Ensure nc is installed on Bastion:**

* Establishes an SSH connection to the bastion host.
* Installs nc (netcat) utility on the bastion host to enable proxying SSH connections to private instances.

### **6. Deploy:**

* Establishes SSH connections to each private instance via the bastion host.
* Stops and removes existing containers if they exist on the private instances.
* Runs the Docker containers for the Java application, Prometheus monitoring tool, and Grafana dashboard:
  + The Java application container is run with port mapping from host port 8081 to container port 8080.
  + Prometheus container is run with port mapping from host port 9090 to container port 9090 and mounts a custom Prometheus configuration file.
  + Grafana container is run with port mapping from host port 3000 to container port 3000 and mounts a persistent volume for Grafana data storage.

### **Additional Notes:**

* SSH connections to private instances are established via the bastion host using ProxyCommand with nc utility.
* Docker containers are stopped and removed before starting new instances to ensure clean deployments.
* Prometheus and Grafana containers are deployed alongside the Java application for monitoring purposes.
* Proper error handling and cleanup mechanisms are implemented to handle failures during deployment.

This pipeline automates the deployment of the Java application and related services onto multiple EC2 instances in a secure manner using SSH tunneling via a bastion host. It ensures consistency, reliability, and ease of management in the deployment process.

What is Git Webhook?  
- Adding a webhook on Git (such as GitHub) allows you to configure automated notifications and trigger actions in external systems, like Jenkins, whenever certain events occur in your Git repository. These events typically include actions like pushing code changes, creating branches, or opening pull requests. This helps streamline your development process by automating tasks and providing real-time notifications about repository events.  
  
  
**Adding Webhook -**

Step 1 - Go to your jenkins and create a new pipeline or configure existing pipeline to choose poll scm and select the option and apply it.  
  
Step2 - Go to manage jenkins -> System and look for your jenkins url. Append ‘github-webhook/’ to your jenkins url and copy it.  
  
Step 3 - Go to your github repo and then settings, click on webhooks to paste the url in the payload url option.   
  
Step 4 - Apply it and when you press on recent deliveries, you should be able to see something similar to the following figure when you test it.

