DEPLOYMENT OF A CONTAINERIZED JAVA BASED WEB APPLICATION USING JENKINS.

BY
EBUKA ONYIA

Introduction

In this project, we deployed a Java application in a containerized environment that is highly secured, available and scalable. In this project, we used docker as our container of choice to deploy our application in both staging and production environment, we also incorporated some security and monitoring measures for our infrastructure and application to make sure they are all in good condition to perform optimally. In addition, we automated a custom solution that would continuously deploy our application when even any change is made.

AIM: The aim of this project is to successfully deploy a java application in a containerized environment that highly available, highly secured and highly scalable.

Tech Stack:

- Terraform: Used for writing the IAC (infrastructure as a code)
- Java: used for writing the application
- AWS cloud: Cloud provider for the IAC
- Bash script: used for configuration of servers.
- Sonarqube: for scanning application code
- Nexus: storing artefact and docker image
- Owasp: dependency scanning tool.

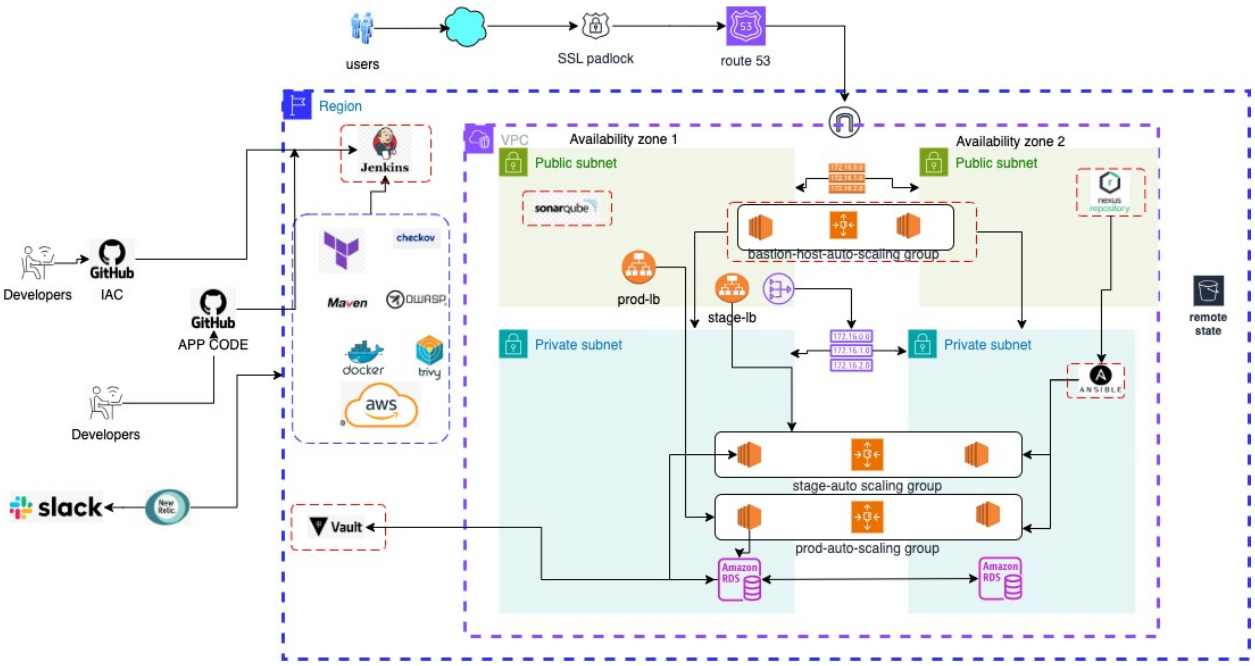
SOURCE CODE LINK
https://github.com/onyiafranklin/autodiscovery-proj infrastructure code link

https://github.com/onyiafranklin/usteam-2 application source code(branch name >>eu-team2-app)

REQUIREMENTS

1. AWS Account
2. Domain name
3. VSCODE(code editor)
4. Github Account

ARCHITECTURAL DIAGRAM



The Architectural diagram above shows all networking components and services combined to successfully deploy our containerized java application using Jenkins pipeline. The following components were used.

1. **AWS Region**: This is a geographical area where AWS hosts its datacentres, and each region consist of multiple availability zones (ie isolated locations within a region). Here, we made use of EU-West-1 region for this project.

2. **VPC**: This is a logical isolated virtual network that we used to lunch all the networking components for this project.
3. **Availability Zone:** This is a physical separate datacentre within an AWS region, it has is own cooling, networking and power components that is isolated from failures in other Availability zones. This means that each AZ fails on its own without affecting the other availability zones. In this project, we made use of EU-west-1a and EU-west-1b.
4. **Private subnets**: This is a network segment without a direct route to the internet. It can't be accessed via the internet and outbound access requires Nat-gatway. In this project, we made use of 2 private subnets on different availability zones to increase redundancy.
5. **Public Subnets:** This is a network segment with access to the internet through the internet gateway. In this project we used 2 public subnets with provisioned on 2 different availability zones.
6. **Internet-gateway:** This is an AWS service that allows communication between your VPC and internet. In this project, we are using it to route traffic to from the public subnet to the internet.
7. **Nat-**gateway: This is a managed AWS service that enables instances in private subnets to initiate outbound internet connections while blocking inbound traffic from the internet
8. **Public route table:** A public route table is a critical networking component that controls traffic routing for public subnets in your VPC.
9. private route table is a set of rules (routes) that determine how network traffic is directed with in the private subnets of a VPC

10. RDS database: This is an aws managed service used to persist users data. We have deployed a multi AZ RDS database for high availability to create room for failover
11. **Stage and production servers:** These are AWS ec2 instances where docker containers are installed that is used to lunch the application. Stage servers are servers where user acceptability test, load performance and smoke test, after which application is deployed on prod servers.
12. **Stage and Prod Load Balancers**: AWS ALBs/NLBs distributing traffic across stage and production environments, ensuring high availability and scalability
13. **Bastion Host**: A secure jump server in a public subnet for SSH access to private instances.
14. **Nexus Server**: Private artifact repository (e.g., Docker, Maven) hosted on EC2 or containers, In this project, we used the nexus to store artefact created by maven and docker image. here we created two registries one for maven hosted repo and the other for docker hosted repo.
15. **SonarQube Servers**: Code quality analysis tools deployed on EC2/containers, integrated into CI/CD pipelines for static code scanning.
16. **Vault Server**: This is HashiCorp Vault for secrets management (API keys, certificates) in private subnets with IAM integration. In this project, we used vault to store the rds credentials (username and password of our database). It was integrated with AWS KMS service which helped us to store the secrets of the vault that would help unseal the vault.
17. **S3 Bucket (Remote State)**: Secured bucket storing Terraform state files with versioning and encryption for IaC management.

18. **Jenkins Server**: CI/CD orchestration server with tools like **Trivy** (vulnerability scanning), **Maven** (Java builds), **Docker** (containerization), **OWASP** (security testing), and **Checkov** (IaC scanning).
19. **Slack**: Notification hub for alerts (e.g., Jenkins builds, security scans) via webhooks or bots .
20. **New Relic**: APM tool monitoring application performance, errors, and infrastructure metrics in prod/stage.
21. **Route53**: Managed DNS service routing traffic to ALBs/NLBs with health checks for failover.
22. **SSL Padlock/AWS ACM**: TLS certificates from ACM enabling HTTPS on ALBs/NLBs, auto renewed and integrated with Route53.
23. **GitHub**: Version control for code repositories, triggering Jenkins pipelines via webhooks for CI/CD automation.

PROCESS FLOW ON INFRASTRUCTURE PROVISIONNING AND APPLICATION DEPLOYMENT

Developers after development, push their code to github, which triggers the Jenkins pipline to deploy the IAC code and application code.
First, before deployment of the IAC code, the Jenkins server and vault server are first provisioned. This is because we Jenkins server would be used to deploy every other infrastructure where our application would be sitting on, this IAC provisioning by Jenkins represents our First pipeline. Also, the vault server was also provisioned alongside with the Jenkins server, this is because the database credential stored in this vault would be used by the database provisioned by Jenkins server during database provisioning.

After the Jenkins and Vault provisioning, the Jenkins server with the help of "JENKINSFILE" containing stages of events that would be executed which would help to provision all other infrastructure where our application would be hosted.

After Infrastructure provisioning by our Jenkins server, the second pipeline with the help of another our application Jenkinsfile houses various steps that would help us deploy our application in a containerized environment would now kick off towards deploying our application.

NETWORK /DATA FLOW DURING USER'S EXPERIENCE

When user visits our application website(www.edenboutique.space) our route53 resolves the url to an ip address. With this , the user gains access to the website, when users request for data from the website, users traffic is encrypted via **TLS/SSL** (using ACM certificate attached to ALB),and sends data t through internet gatway to the load balancers with data still encrypted, whereby the loadbalancers sends down decrypted information down to our docker servers in private subnet hosting the application.in same manner, data is sent back to the user in same manner through the Nat-gateway for consumption.

STEP BY STEP PROCEDURE TO SUCCESSFULLY PROVISION OUR INFRASTRUCTURE, DEPLOY OUR APPLICATION, PERSIST DATA ON DATABASE AND MONITORING USING NEWRELIC.

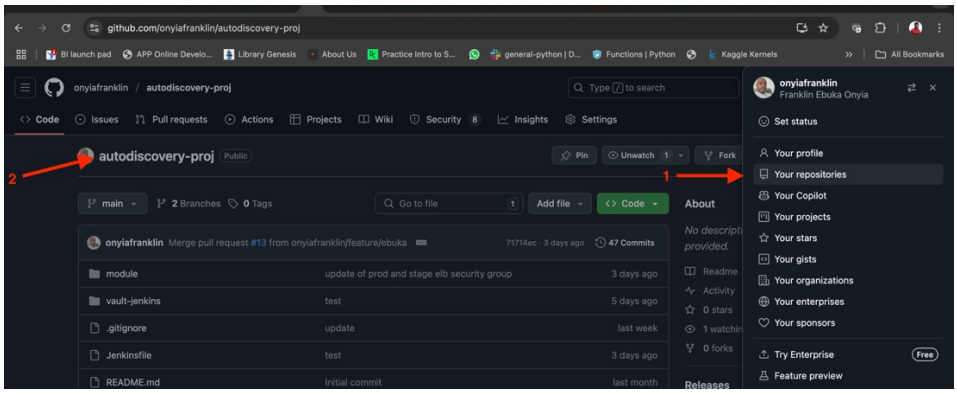
The steps below would lead us on the procedure on how to deploy our application successfully starting from infrastructure provisioning using terraform on vscode using a modularized templete for various infrastructure to writing Jenkins file that would deploy our infrastructure down to the application docker file where the application is been containerized to the Jenkins file that deploy the containerized application, then to the connection of our application to persist data on our RDS and monitoring procedure for our application and infrastructure using Newrelic. The steps are listed below

a. Create a repo on Github and clone the repo on your local machine and setup file structure for all your code
b. Create script that would create and destroy remote state bucket
c. Vault and Jenkins server provisioning
d. Modular code explanations for nexus ,sonarqube, database, stage environment, prod environment, database.
e. Ansible code explanation
f. Infrastructure jenkinsfile explanation
g. Infrastructure provisioning setup for s3,vault and jenkins
h. Infrastructure setup and provisioning using Jenkins pipeline
i. Application jenkinsfile explanation
j. Jenkins pipeline setup for application and application deployment
k. Persisting our data to our database
l. Change on application pipeline and redeployment to view change
m. Monitoring using Newrelic

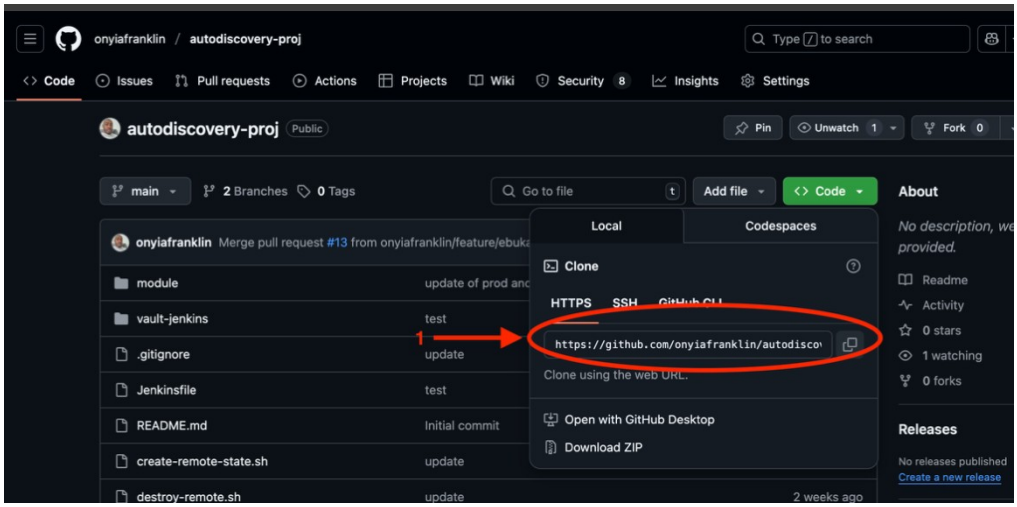
A. CREATE A REPOSITORY ON YOUR GITHUB AND CLONE ON YOUR LOCAL MACHINE

Here we are to create a repository account for collaboration and storing and working on different versions of code.

1. Go to www.github.com and create an account, there after create a repository.



2. Copy the url to clone the repository on your local machine, go to your terminal and type “git clone <url>”

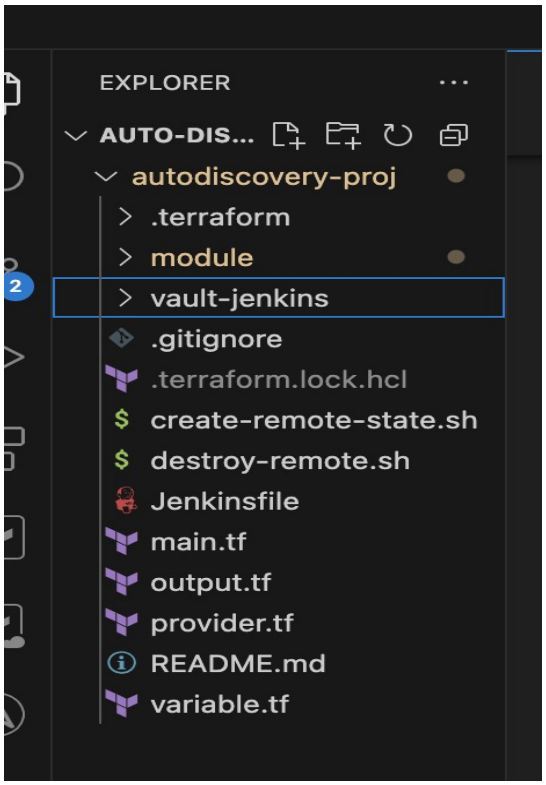


3. After cloning on your terminal, Create modules and file for the project

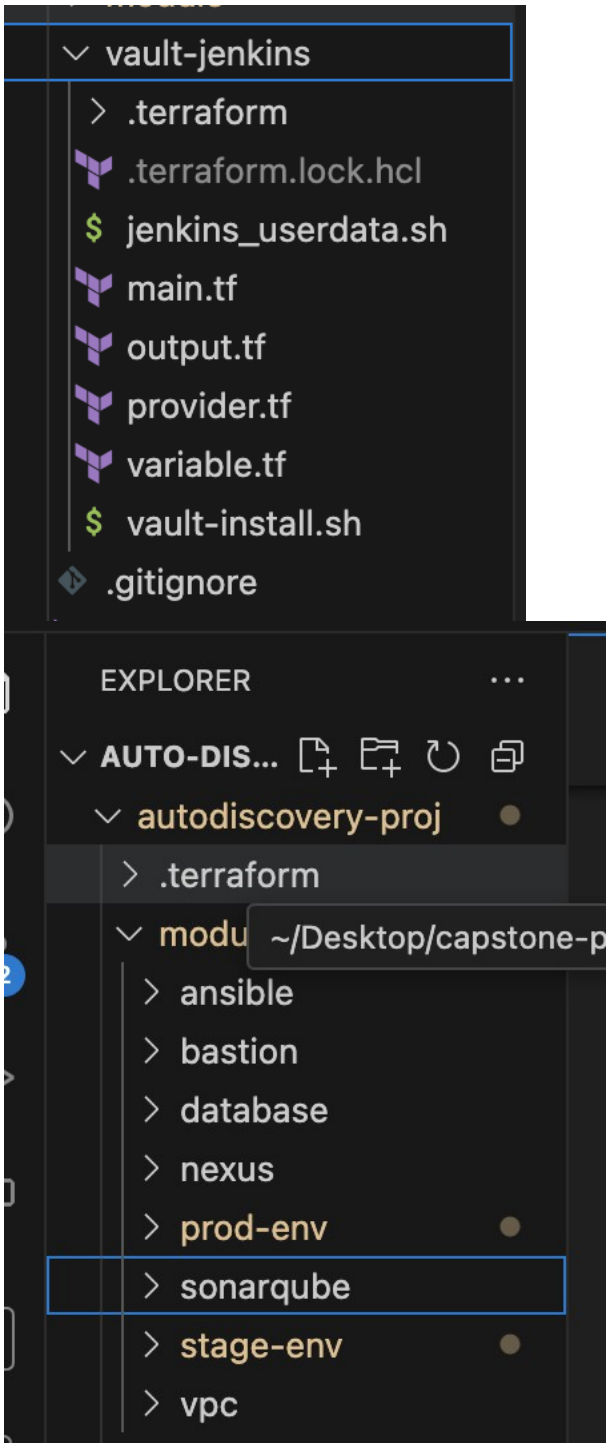
   Create the folders and files show below in the cloned repository folder.this includes

   the vault-jenkins folder where we would provision the vault and Jenkins server

   Create the root main,output,variable .tf files



4. Create modules folder where we would provision all the modules for our sonarqube,nexus, prod ,stage server, and database modules.


vault-jenkins
.terraform
.terraform.lock.hcl
jenkins_userdata.sh
main.tf
output.tf
provider.tf
variable.tf
vault-install.sh
.gitignore
EXPLORER
AUTO-DIS...
autodiscovery-proj
.terraform
modu
~/Desktop/capstone-p
ansible
bastion
database
nexus
prod-env
sonarqube
stage-env
vpc

B. CREATE SCRIPT THAT WOULD CREATE AND DESTROY REMOTE STATE BUCKET.

Here we created a script called “create remote state” the main function of this script is to create an s3 bucket that would be used to store our terraform state file. This is because you won’t be the only one working on this project, you have other team members working along with you, it helps to preserver the state of your terraform code to prevent simultaneous change at same time when both teams are working on the code. So therefore, it locks the state when the first member of the team is working on the project and when the first person completes its work, the second team member can now pull the latest terraform state file and works it as well.

CREATE S3 BUCKET

The code snipet helps us to create bucket called “bucket-pet-adoption” in eu-west-2 region. The final part of the code also helps us to create Jenkins and vault server.

```
#!/bin/bash
set -euo pipefail  # Enable strict error handling

# Set Variables
BUCKET_NAME="bucket-pet-adoption"
AWS_REGION="eu-west-2"
PROFILE="bukky_int"

# Function to handle errors
handle_error() {
   echo "□ Error: $1"
   exit 1
}

# Create S3 Bucket
echo "□ Creating S3 bucket: $BUCKET_NAME..."
if aws s3api head-bucket --bucket "$BUCKET_NAME" --region "$AWS_REGION" --profile "$PROFILE" 2>/dev/null;
then
   echo "⚠  Bucket '$BUCKET_NAME' already exists. Skipping creation."
else
   if ! aws s3api create-bucket --bucket "$BUCKET_NAME" --region "$AWS_REGION" --profile "$PROFILE" --
create-bucket-configuration LocationConstraint="$AWS_REGION"; then
      handle_error "Failed to create S3 bucket '$BUCKET_NAME'."
   fi
   echo "□ S3 bucket '$BUCKET_NAME' created successfully."
fi

# # Enable versioning
```

```
echo "🔄 Enabling versioning for S3 bucket..."
if ! aws s3api put-bucket-versioning --bucket "$BUCKET_NAME" --versioning-configuration Status=Enabled --
region "$AWS_REGION" --profile "$PROFILE"; then
    handle_error "Failed to enable versioning for S3 bucket '$BUCKET_NAME'."
fi
echo "✅ Versioning enabled successfully."

echo "🎉 S3 Remote State Management Setup Complete!"
echo "🌍 S3 Bucket: $BUCKET_NAME"

# provision the vault and jenkins server
echo "🚀 Provisioning Vault and Jenkins server..."
cd ./vault-jenkins
terraform init
terraform fmt --recursive
terraform validate
terraform apply -auto-approve
terraform output
echo "✅ Vault and Jenkins server provisioned successfully."
```

DESTROY S3 BUCKET

This the major role of this script is to destroy the s3 bucket created as well as destroy the jenkins and vault server

```
#!/bin/bash
set -euo pipefail  # Enable strict error handling

# Set Variables
BUCKET_NAME="bucket-pet-adoption"
AWS_REGION="eu-west-2"
PROFILE="bukky_int"

# destroy vault and jenkins server
cd ./vault-jenkins
terraform destroy -auto-approve
terraform output

# Function to handle errors
handle_error() {
    echo "❌ Error: $1"
    exit 1
}

# Ensure AWS CLI is installed
if ! command -v aws &>/dev/null; then
    handle_error "AWS CLI is not installed. Please install it and retry."
fi
```

```
# Check if S3 bucket exists before attempting deletion
if aws s3api head-bucket --bucket "$BUCKET_NAME" --region "$AWS_REGION" --
profile "$PROFILE" 2>/dev/null; then
    echo "🗑 Deleting all objects from S3 bucket: $BUCKET_NAME..."

    # Delete all versions of objects
    versions=$(aws s3api list-object-versions --bucket "$BUCKET_NAME" --
query='{Objects: Versions[].{Key:Key,VersionId:VersionId}}' --output=json --
profile "$PROFILE")
    if [ "$versions" != '{"Objects":[]}' ]; then
        echo "$versions" > delete.json
        if ! aws s3api delete-objects --bucket "$BUCKET_NAME" --delete
file://delete.json --profile "$PROFILE"; then
            handle_error "Failed to delete versioned objects from S3 bucket
'$BUCKET_NAME'."
        fi
        rm -f delete.json
    fi

    # Delete all delete markers
    delete_markers=$(aws s3api list-object-versions --bucket "$BUCKET_NAME"
--query='{Objects: DeleteMarkers[].{Key:Key,VersionId:VersionId}}' --
output=json --profile "$PROFILE")
    if [ "$delete_markers" != '{"Objects":[]}' ]; then
        echo "$delete_markers" > delete_markers.json
        if ! aws s3api delete-objects --bucket "$BUCKET_NAME" --delete
file://delete_markers.json --profile "$PROFILE"; then
            handle_error "Failed to delete delete markers from S3 bucket
'$BUCKET_NAME'."
        fi
        rm -f delete_markers.json
    fi

    echo "✅ All objects, including versioned ones, deleted from S3 bucket."

    echo "🗑 Deleting S3 bucket: $BUCKET_NAME..."
    if ! aws s3api delete-bucket --bucket "$BUCKET_NAME" --region
"$AWS_REGION" --profile "$PROFILE"; then
        handle_error "Failed to delete S3 bucket '$BUCKET_NAME'."
    fi
    echo "✅ S3 bucket '$BUCKET_NAME' deleted successfully."
else
    echo "⚠️  S3 bucket '$BUCKET_NAME' does not exist. Skipping deletion."
fi

echo "🎉 S3 Bucket Cleanup Complete!"
```

C. VAULT AND JENKINS SERVER PROVISIONING

The vault and Jenkins server are provisioned immediately after the bucket is created. This is because the Jenkins server needs to be up and running because it would be used to provision the infrastructure where our application would be running.Also the vault server needs to be up and running because the database that would be provisioned by the jenkins needs to make an API call to the vault server to use the credentials stored in the vault server.

For vault

The resource block “resource "aws_kms_key" "kms-key" below shows the creation of KMS key that would be used to manage the vault secrets.
This followed by creation of IAM role and policy using the “resource "aws_iam_role" "vault_role" and resource "aws_iam_role_policy" "vault_kms_access" resource block “that would enable the vault server to perform some encryption and decryption role.

Next is the resource "aws_iam_instance_profile" "vault-profile" whose role be attached to the vault server during creation. This is followed by “resource "aws_security_group" "vault_sg"“ which is opened at port 22 for ssh, port 80 for http and port 8200 is default port for hasicorp vault.
Also we have created a network /elastic load balancer resource "aws_elb" "elb-vault" for the vault server opened on port 443 so it can interface with the external environment and receive request,it was also opened on instance port 8200 to interact with the vault server itself. Furthermore, we created a security group for the vault elastic load balancer resource "aws_security_group" "elb-vault-sg" with ingress at port 443 and egress at anyport. Vaultserver with resource block resource "aws_instance" "vault_server" is not left out as it was created in the default VPC here for testing purpose. Also, we created an AWS certificate manager resource resource "aws_acm_certificate" "acm-cert" for our domain and subdomain together with its record. after the certificate has been issued, we moved ahead with creation of records for our vault server on resource "aws_route53_record" "vault-record" .

For Jenkins,

This started with the creation of IAM role for jenkins server resource "aws_iam_role" "jenkins-role" in which administrator role resource "aws_iam_role_policy_attachment" "jenkins-role-attachment"is attached to the role, after which iam instance profile (resource "aws_iam_instance_profile" "jenkins-profile")is attached to the jenkins server during creation

Next is the creation of Security group(resource "aws_security_group" "jenkins_sg") for for jenkins server which is opened on opened at port 22 for ssh, port 443 for https and port 8080 is default port for jenkins
Also we have created a network /elasticload balancer resource (resource "aws_elb" "elb_jenkins") for the Jenkins server opened on port 443 so it can interface with the external environment and

receive request,it was also opened on instance port 8080 to interact with the jenkins server itself. Furthermore, we created a security group for the jenkins elastic load balancer resource (resource "aws_security_group" "jenkins-elb-sg") with ingress at port 443 and egress at anyport. Jenkins server with resource block resource resource ("aws_instance" "jenkins-server") is not left out as it was created in the default VPC here for testing purpose. Also, we created an AWS certificate manager resource resource "aws_acm_certificate" "acm-cert" for our domain and subdomain together with its record. after the certificate has been issued, we moved ahead with creation of records for our jenkins server on resource (resource "aws_route53_record" "jenkins-record").

```
main.tf
locals {
  name = "auto-discov"
}
#Creating kms key
resource "aws_kms_key" "kms-key" {
  description             = "${local.name}-vault-kms-key" // the kms key
  deletion_window_in_days = 30                            // 30 days
  enable_key_rotation     = true                          // true
}
# alias of the kms key
resource "aws_kms_alias" "kms-key" {
  name          = "alias/${local.name}-kms-key"
  target_key_id = aws_kms_key.kms-key.key_id
}

resource "aws_iam_role" "vault_role" {
  name = "${local.name}-vault-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          Service = "ec2.amazonaws.com"
        },
        Action = "sts:AssumeRole"
      }
    ]
  })
}


resource "aws_iam_role_policy" "vault_kms_access" {
  name = "${local.name}-vault_kms-access"
  role = aws_iam_role.vault_role.id

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
```

```
        Action = [
          "kms:Encrypt",
          "kms:Decrypt",
          "kms:ReEncrypt*",
          "kms:GenerateDataKey*",
          "kms:DescribeKey"
        ],
        Resource = "${aws_kms_key.kms-key.arn}"
      }
    ]
  })
}
resource "aws_iam_instance_profile" "vault-profile" {
  name = "${local.name}-vault-profile"
  role = aws_iam_role.vault_role.name
}

# Create Vault Security Group
resource "aws_security_group" "vault_sg" {
  name        = "vault-sg"
  description = "Allow SSH, HTTP, HTTPS, and Vault UI/API"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 8200
    to_port     = 8200
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
# Ubuntu AMI lookup
```

```
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}
resource "aws_instance" "vault_server" {
  ami                  = data.aws_ami.ubuntu.id # Ubuntu in eu-west-2
  instance_type        = "t2.medium"
  key_name             = aws_key_pair.public_key.key_name
  security_groups      = [aws_security_group.vault_sg.name]
  iam_instance_profile = aws_iam_instance_profile.vault-profile.id
  user_data = templatefile("./vault-install.sh", {
    var1 = "eu-west-2",
    var2 = aws_kms_key.kms-key.id
  })

  tags = {
    Name = "${local.name}-VaultServer"
  }
}

# Create keypair resource
resource "tls_private_key" "keypair" {
  algorithm = "RSA"
  rsa_bits  = 4096
}
resource "local_file" "private_key" {
  content         = tls_private_key.keypair.private_key_pem
  filename        = "${local.name}-key.pem"
  file_permission = "400"
}
resource "aws_key_pair" "public_key" {
  key_name   = "${local.name}-key"
  public_key = tls_private_key.keypair.public_key_openssh
}
# Data source to get the latest RedHat AMI
data "aws_ami" "redhat" {
  most_recent = true
  owners      = ["309956199498"] # RedHat's owner ID
  filter {
    name   = "name"
    values = ["RHEL-9*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
```

```
  }
}
resource "aws_instance" "jenkins-server" {
  ami                         = data.aws_ami.redhat.id # redhat in eu-west-2
  instance_type               = "t2.medium"
  key_name                    = aws_key_pair.public_key.id
  associate_public_ip_address = true
  vpc_security_group_ids      = [aws_security_group.jenkins_sg.id]
  iam_instance_profile        = aws_iam_instance_profile.jenkins-profile.id
  root_block_device {
    volume_size = 30    # Size in GB
    volume_type = "gp3" # General Purpose SSD (recommended)
    encrypted   = true  # Enable encryption (best practice)
  }
  user_data = templatefile("./jenkins_userdata.sh", {
    nr-key    = "NRAK-4FNJBSGOTULJ4XCZW4P2JOMPOKY",
    nr-acc-id = 6496342
  })

  tags = {
    Name = "${local.name}-jenkins-server"
  }
}
# Create IAM role for Jenkins
resource "aws_iam_role" "jenkins-role" {
  name = "${local.name}-jenkins-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          Service = "ec2.amazonaws.com"
        },
        Action = "sts:AssumeRole"
      }
    ]
  })
}
# Attach the policy to the role
resource "aws_iam_role_policy_attachment" "jenkins-role-attachment" {
  role       = aws_iam_role.jenkins-role.name
  policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
}
# Attach the policy to the role
resource "aws_iam_instance_profile" "jenkins-profile" {
  name = "${local.name}-jenkins-profile"
  role = aws_iam_role.jenkins-role.name
}

# Create jenkins security group
resource "aws_security_group" "jenkins_sg" {
  name        = "${local.name}-jenkins-sg"
  description = "Allow SSH and HTTPS"

  ingress {
```

```
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

#create a time sleep resource that allow terraform to wait till vault server
is ready
resource "time_sleep" "wait_3_min" {
  depends_on      = [aws_instance.vault_server]
  create_duration = "180s"
}

#create null resource to fetch vault token
resource "null_resource" "fetch_token" {
  depends_on = [time_sleep.wait_3_min]
  # create terraform provisioner to help fetch token file from the vault
server
  provisioner "local-exec" {
    command = "scp -o StrictHostKeyChecking=no -i ./${local.name}-key.pem
ubuntu@${aws_instance.vault_server.public_ip}:/home/ubuntu/token.txt ."
  }

  #  provisioner "locai-exec" {
  #    interpreter=["bash", "-c"]
  #    command= "sed -i '' \"s/token = \\\".*\\\"/token = \\\"$(cat
./token.txt)\\\"/\" ../provider.t"
  #  }
  provisioner "local-exec" {
    when    = destroy
    command = "rm -f ./token.txt"
  }
}
# Create ACM certificate with DNS validation
resource "aws_acm_certificate" "acm-cert" {
  domain_name               = var.domain
```

```
  subject_alternative_names = ["*.${var.domain}"]
  validation_method         = "DNS"
  lifecycle {
    create_before_destroy = true
  }

  tags = {
    Name = "${local.name}-acm-cert"
  }
}

data "aws_route53_zone" "acp-zone" {
  name         = var.domain
  private_zone = false
}

# Fetch DNS Validation Records for ACM Certificate
resource "aws_route53_record" "acm_validation_record" {
  for_each = {
    for dvo in aws_acm_certificate.acm-cert.domain_validation_options :
dvo.domain_name => {
      name   = dvo.resource_record_name
      record = dvo.resource_record_value
      type   = dvo.resource_record_type
    }
  }

  # Create DNS Validation Record for ACM Certificate
  zone_id         = data.aws_route53_zone.acp-zone.zone_id
  allow_overwrite = true
  name            = each.value.name
  type            = each.value.type
  ttl             = 60
  records         = [each.value.record]
  depends_on      = [aws_acm_certificate.acm-cert]
}

# Validate the ACM Certificate after DNS Record Creation
resource "aws_acm_certificate_validation" "team2_cert_validation" {
  certificate_arn         = aws_acm_certificate.acm-cert.arn
  validation_record_fqdns = [for record in
aws_route53_record.acm_validation_record : record.fqdn]
  depends_on              = [aws_acm_certificate.acm-cert]
}

#Create Security Group for Vault Elastic Load Balancer
resource "aws_security_group" "elb-vault-sg" {
  name        = "elb-vault-sg"
  description = "Allow HTTPS"

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

```
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Create load balancer for Vault Server
resource "aws_elb" "elb-vault" {
  name               = "vault-elb"
  availability_zones = ["eu-west-2a", "eu-west-2b"]

  listener {
    instance_port      = 8200
    instance_protocol  = "http"
    lb_port            = 443
    lb_protocol        = "https"
    ssl_certificate_id = aws_acm_certificate.acm-cert.arn
  }

  health_check {
    healthy_threshold   = 2
    unhealthy_threshold = 2
    timeout             = 3
    target              = "TCP:8200"
    interval            = 30
  }

  instances                   = [aws_instance.vault_server.id]
  cross_zone_load_balancing   = true
  idle_timeout                = 400
  connection_draining         = true
  connection_draining_timeout = 400

  tags = {
    Name = "${local.name}-elb-vault"
  }
}

# Create Route 53 Hosted Zone
data "aws_route53_zone" "vault-zone" {
  name         = var.domain
  private_zone = false
}

# Create Route 53 A Record for Vault Server
resource "aws_route53_record" "vault-record" {
  zone_id = data.aws_route53_zone.vault-zone.zone_id
  name    = "vault.${var.domain}"
  type    = "A"
  alias {
    name                   = aws_elb.elb-vault.dns_name
    zone_id                = aws_elb.elb-vault.zone_id
    evaluate_target_health = true
  }
}
```

```
# Create elastic Load Balancer for Jenkins
resource "aws_elb" "elb_jenkins" {
  name               = "elb-jenkins"
  security_groups    = [aws_security_group.jenkins-elb-sg.id]
  availability_zones = ["eu-west-2a", "eu-west-2b"]
  listener {
    instance_port      = 8080
    instance_protocol  = "HTTP"
    lb_port            = 443
    lb_protocol        = "HTTPS"
    ssl_certificate_id = aws_acm_certificate.acm-cert.id
  }
  health_check {
    healthy_threshold   = 3
    unhealthy_threshold = 2
    interval            = 30
    timeout             = 5
    target              = "TCP:8080"
  }
  instances                   = [aws_instance.jenkins-server.id]
  cross_zone_load_balancing   = true
  idle_timeout                = 400
  connection_draining         = true
  connection_draining_timeout = 400
  tags = {
    Name = "${local.name}-jenkins-server"
  }
}
# Create Security group for the jenkins elb
#
resource "aws_security_group" "jenkins-elb-sg" {
  name        = "${local.name}-jenkins-elb-sg"
  description = "Allow HTTPS"

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Create Route 53 record for jenkins server
resource "aws_route53_record" "jenkins-record" {
  zone_id = data.aws_route53_zone.acp-zone.zone_id
  name    = "jenkins.${var.domain}"
  type    = "A"
  alias {
    name                   = aws_elb.elb_jenkins.dns_name
    zone_id                = aws_elb.elb_jenkins.zone_id
    evaluate_target_health = true
```

```
  }
}
```

userdata for Jenkins server
here we installed all dependencies for our jenkins server. some of them include maven,wget,git, jenkins, docker,trivy,awscli, newrelic

```
#!/bin/bash
sudo yum update -y
sudo yum install wget -y
sudo yum install maven -y
sudo yum install git pip -y
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-
stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum upgrade -y
sudo yum install java-17-openjdk -y
sudo yum install jenkins -y
sudo sed -i 's/^User=jenkins/User=root/'
/usr/lib/systemd/system/jenkins.service
sudo systemctl daemon-reload
sudo systemctl start jenkins
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo usermod -aG jenkins ec2-user
# Install trivy for container scanning
RELEASE_VERSION=$(grep -Po '(?<=VERSION_ID=")[0-9]' /etc/os-release)
cat << EOT | sudo tee -a /etc/yum.repos.d/trivy.repo
[trivy]
name=Trivy repository
baseurl=https://aquasecurity.github.io/trivy-repo/rpm/releases/
$RELEASE_VERSION/\$basearch/
gpgcheck=0
enabled=1
EOT
sudo yum -y update
sudo yum -y install trivy
# installing docker
sudo yum install -y yum-utils
sudo yum config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install docker-ce -y
sudo systemctl start docker
sudo systemctl enable docker

# Installing awscli
sudo yum install unzip -y
sudo curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
sudo unzip awscliv2.zip
sudo ./aws/install
sudo ln -svf /usr/local/bin/aws /usr/bin/aws

#  install newrelic agent
```

```
 curl -Ls
https://download.newrelic.com/install/newrelic-cli/scipts/install.sh | bash
&& sudo NEW_RELIC_API_KEY="${nr-key}" NEW_RELIC_ACCOUNT_ID="${nr-acc-id}"
NEW_RELIC_REGION=EU /usr/local/bin/newrelic install -y
sudo hostnamectl set-hostname jenkins
```

Userdada for vault server

here we intalled all dependencies such as the consul(database of the vault server), we also installed the vault and initialise the vault to get the secrects that would be used to unseal the vault. vault has 5 secrects stored init but we require only 3 to unseal the vault, so with the help of kms(key management service), we are able to manage these secrects in kms .

```
#!/bin/bash

# Update package repositories
sudo apt update

# Download and install Consul
sudo wget
https://releases.hashicorp.com/consul/1.7.3/consul_1.7.3_linux_amd64.zip
sudo apt install unzip -y
sudo unzip consul_1.7.3_linux_amd64.zip
sudo mv consul /usr/bin/

# Create a Consul systemd service
sudo cat <<EOT>> /etc/systemd/system/consul.service
[Unit]
Description=Consul
Documentation=https://www.consul.io/

[Service]
ExecStart=/usr/bin/consul agent -server -ui -data-dir=/temp/consul -
bootstrap-expect=1 -node=vault -bind=$(hostname -i)
-config-dir=/etc/consul.d/
ExecReload=/bin/kill -HUP $MAINPID
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOT

# Create Consul configuration directory and UI settings
sudo mkdir /etc/consul.d
sudo cat <<EOT>> /etc/consul.d/ui.json
{
    "addresses":{
    "http": "0.0.0.0"
    }
}
EOT
```

```
# Reload systemd, start, and enable Consul service
sudo systemctl daemon-reload
sudo systemctl start consul
sudo systemctl enable consul

# Download and install Vault
sudo apt update
sudo wget
https://releases.hashicorp.com/vault/1.5.0/vault_1.5.0_linux_amd64.zip
sudo unzip vault_1.5.0_linux_amd64.zip
sudo mv vault /usr/bin/

# Create Vault configuration file
sudo mkdir /etc/vault/
sudo cat <<EOT>> /etc/vault/config.hcl
storage "consul" {
    address = "127.0.0.1:8500"
    path    = "vault/"
}

listener "tcp" {
    address        = "0.0.0.0:8200"
    tls_disable    = 1
}

seal "awskms" {
    region     = "${var1}"
    kms_key_id = "${var2}"
}
ui = true  #to acess/visualize vault thru the browser
EOT

# Create Vault systemd service
#creating a service file for vault
sudo cat <<EOT>> /etc/systemd/system/vault.service
[Unit]
Description=Vault
Documentation=https://www.vault.io/

[Service]
ExecStart=/usr/bin/vault server -config=/etc/vault/config.hcl
ExecReload=/bin/kill -HUP $MAINPID
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOT

# Reload systemd, start, and enable Vault service
sudo systemctl daemon-reload
export VAULT_ADDR="http://localhost:8200"
cat <<EOT > /etc/profile.d/vault.sh
export VAULT_ADDR="http://localhost:8200"
export VAULT_SKIP_VERIFY=true
EOT
vault -autocomplete-install
```

```
complete -C /usr/bin/vault vault

# Notify once provisioned
echo "Vault server provisioned successfully."
# Start Vault service
sudo systemctl start vault
sudo systemctl enable vault
sleep 20
# #Set vault token/secret username and password
touch /home/ubuntu/output.txt
vault operator init > /home/ubuntu/output.txt
grep -o 's\.[A-Za-z0-9]\{24\}' /home/ubuntu/output.txt >
/home/ubuntu/token.txt
token_content=$(</home/ubuntu/token.txt)
vault login $token_content
vault secrets enable -path=secret/ kv #directory to store secrets on the
vault server
vault kv put secret/database username=petclinic password=petclinic
# Set hostname to Vault
sudo hostnamectl set-hostname Vault
```

D. MODULAR CODE EXPLANATIONS FOR NEXUS ,SONARQUBE, DATABASE, STAGE ENVIRONMENT, PROD ENVIRONMENT, DATABASE.

VPC Module

This is a virtual private cloud set apart to host all networking components for our infrastructure. here we have deployed our custom VPC, interner gateway, private and public subnets,public and private route tables in ADDITION TO THE NATgateway. we also created a keypair that would be used during our server creation. this keypair would enable us access whenever we want to ssh into the server . This was created using the RSA algorithm

```
main.tf
#Creating the vpc
resource "aws_vpc" "vpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "${var.name}-vpc"
  }
}

# create public subnet 1
resource "aws_subnet" "pub_sub1" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = var.az1

  tags = {
    Name = "${var.name}-pub_sub1"
  }
```

```
}

# create public subnet 2
resource "aws_subnet" "pub_sub2" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.2.0/24"
  availability_zone = var.az2

  tags = {
    Name = "${var.name}-pub_sub2"
  }
}

# create private subnet 1
resource "aws_subnet" "pri_sub1" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.3.0/24"
  availability_zone = var.az1

  tags = {
    Name = "${var.name}-pri_sub1"
  }
}

# create private subnet 2
resource "aws_subnet" "pri_sub2" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.4.0/24"
  availability_zone = var.az2

  tags = {
    Name = "${var.name}-pri_sub2"
  }
}

# create internet gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.vpc.id

  tags = {
    Name = "${var.name}-igw"
  }
}

# create elastic ip for NAT gateway
resource "aws_eip" "eip" {
  domain = "vpc"

  tags = {
    Name = "${var.name}-eip"
  }
}

# create NAT gateway
resource "aws_nat_gateway" "ngw" {
  allocation_id = aws_eip.eip.id
  subnet_id     = aws_subnet.pub_sub1.id
```

```
  tags = {
    Name = "${var.name}-ngw"
  }
  depends_on = [aws_eip.eip] # Ensuring the EIP is created first
}


# Create route table for public subnets
resource "aws_route_table" "pub_rt" {
  vpc_id = aws_vpc.vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
  tags = {
    Name = "${var.name}-pub_rt"
  }
}

# Create route table for private subnets
resource "aws_route_table" "pri_rt" {
  vpc_id = aws_vpc.vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_nat_gateway.ngw.id
  }
  tags = {
    Name = "${var.name}-pri_rt"
  }
}

# Creating route table association for public_subnet_1
resource "aws_route_table_association" "ass-public_subnet_1" {
  subnet_id      = aws_subnet.pub_sub1.id
  route_table_id = aws_route_table.pub_rt.id
}

# Creating route table association for public_subnet_2
resource "aws_route_table_association" "ass-public_subnet_2" {
  subnet_id      = aws_subnet.pub_sub2.id
  route_table_id = aws_route_table.pub_rt.id
}

# Creating route table association for private_subnet_1
resource "aws_route_table_association" "ass-private_subnet_1" {
  subnet_id      = aws_subnet.pri_sub1.id
  route_table_id = aws_route_table.pri_rt.id
}

# Creating route table association for private_subnet_2
resource "aws_route_table_association" "ass-private_subnet_2" {
  subnet_id      = aws_subnet.pri_sub2.id
  route_table_id = aws_route_table.pri_rt.id
}

#creating keypair RSA key of size 4096 bits
```

```
resource "tls_private_key" "key" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

# Creating private key
resource "local_file" "private-key" {
  content         = tls_private_key.key.private_key_pem
  filename        = "${var.name}-key.pem"
  file_permission = 440
}

# Creating public key
resource "aws_key_pair" "public-key" {
  key_name   = "${var.name}-infra-key"
  public_key = tls_private_key.key.public_key_openssh
}
```

Nexus Module

The nexus server serves as our online repository manager. Over here we would be creating two registries, first repository would be maven hosted repo where we would store the artefacts created by maven and the second registry would be Docker hosted where we would store the docker image of our java based application.
In this code, we created a security group resource block (resource "aws_security_group" "nexus-sg")for our nexus server which is opened on port 22 for only bastion host , other opened port include port 8081 for our maven hosted repository and port 8085 for docker hosted repository. next is to create the nexus server (resource "aws_instance" "nexus")on the public subnet together with the userdatascript that helps to install all dependencies the nexus server uses, also we added an elastic load balancers resource block (resource "aws_security_group" "nexus-elb-sg") and its security group(resource "aws_security_group" "nexus-elb-sg") . The nexus elb security group is opened on port 443 to interface with internet securely, while the load balancer listens on port 8081.Finally we created a record for our nexus server using the (resource "aws_route53_record" "nexus-record") block

```
# Creating Nexus Security group
resource "aws_security_group" "nexus-sg" {
  name        = "${var.name}-nexus-sg"
  description = "Allow SSH, HTTP, HTTPS, nexus and docker "
  vpc_id      = var.vpc

  ingress {
    from_port       = 22
    to_port         = 22
    protocol        = "tcp"
    security_groups = [var.baston-sg]
  }
  ingress {
    from_port   = 8081
    to_port     = 8081
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

```
  ingress {
    from_port   = 8085
    to_port     = 8085
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.name}-nexus-sg"
  }
}

# Data source to get the latest RedHat AMI
data "aws_ami" "redhat" {
  most_recent = true
  owners      = ["309956199498"] # RedHat's owner ID
  filter {
    name   = "name"
    values = ["RHEL-9.4.0_HVM-20240605-x86_64-82-Hourly2-GP3"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}

resource "aws_instance" "nexus" {
  ami                         = data.aws_ami.redhat.id
  # ami                         = "ami-07d4917b6f95f5c2a" # Red Hat
Enterprise Linux
  instance_type               = "t2.medium"
  subnet_id                   = var.subnet-id
  vpc_security_group_ids      = [aws_security_group.nexus-sg.id]
  key_name                    = var.keypair
  associate_public_ip_address = true
  user_data = local.userdata
  root_block_device {
    volume_size = 30    # Size in GB
    volume_type = "gp3" # General Purpose SSD (recommended)
    encrypted   = true  # Enable encryption (best practice)
  }

  tags = {
    Name = "${var.name}-nexus-server"
  }
```

```
}
# Create Security group for the nexus elb
resource "aws_security_group" "nexus-elb-sg" {
  name        = "${var.name}-nexus-elb-sg"
  description = "Allow HTTPS"
  vpc_id      = var.vpc

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "TCP"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-nexus-elb-sg"
  }
}
# Create elastic Load Balancer for nexus
resource "aws_elb" "elb_nexus" {
  name            = "${var.name}-nexus-elb"
  subnets         = [var.subnet1_id, var.subnet2_id]
  security_groups = [aws_security_group.nexus-elb-sg.id]

  listener {
    instance_port      = 8081
    instance_protocol  = "HTTP"
    lb_port            = 443
    lb_protocol        = "HTTPS"
    ssl_certificate_id = var.acm_certificate_arn
  }
  health_check {
    healthy_threshold   = 3
    unhealthy_threshold = 5
    interval            = 30
    timeout             = 5
    target              = "TCP:8081"
  }
  instances                   = [aws_instance.nexus.id]
  cross_zone_load_balancing   = true
  idle_timeout                = 400
  connection_draining         = true
  connection_draining_timeout = 400
  tags = {
    Name = "${var.name}-nexus-elb"
  }
}

# Create Route 53 record for bastion host
data "aws_route53_zone" "acp-zone" {
  name         = var.domain
  private_zone = false
```

```
}

# Create Route 53 record for nexus server
resource "aws_route53_record" "nexus-record" {
  zone_id = data.aws_route53_zone.acp-zone.zone_id
  name    = "nexus.${var.domain}"
  type    = "A"
  alias {
    name                   = aws_elb.elb_nexus.dns_name
    zone_id                = aws_elb.elb_nexus.zone_id
    evaluate_target_health = true
  }
}

resource "null_resource" "update_jenkins" {
  depends_on = [ aws_instance.nexus ]
  provisioner "local-exec" {
    command = <<EOF
#!/bin/bash
sudo cat <<EOT>> /etc/docker/daemon.json
{
  "insecure-registries" : ["${aws_instance.nexus.public_ip}:8085"]
}
EOT
EOF
    interpreter = ["bash", "-c"]
  }
}
```

Userdata script for the nexus server helps to install all dependencies for our nexus server

```
locals {
userdata = <<EOF
#!/bin/bash
sudo yum update -y
sudo yum install wget -y
sudo yum install java-1.8.0-openjdk.x86_64 -y
sudo mkdir /app && cd /app
sudo wget http://download.sonatype.com/nexus/3/nexus-3.23.0-03-unix.tar.gz
sudo tar -xvf nexus-3.23.0-03-unix.tar.gz
sudo mv nexus-3.23.0-03 nexus
sudo adduser nexus
sudo chown -R nexus:nexus /app/nexus
sudo chown -R nexus:nexus /app/sonatype-work
sudo cat <<EOT> /app/nexus/bin/nexus.rc
run_as_user="nexus"
EOT
sed -i '2s/-Xms2703m/-Xms512m/' /app/nexus/bin/nexus.vmoptions
sed -i '3s/-Xmx2703m/-Xmx512m/' /app/nexus/bin/nexus.vmoptions
sed -i '4s/-XX:MaxDirectMemorySize=2703m/-XX:MaxDirectMemorySize=512m/'
/app/nexus/bin/nexus.vmoptions
sudo touch /etc/systemd/system/nexus.service
sudo cat <<EOT> /etc/systemd/system/nexus.service
[Unit]
Description=nexus service
After=network.target
```

```
[Service]
Type=forking
LimitNOFILE=65536
User=nexus
Group=nexus
ExecStart=/app/nexus/bin/nexus start
ExecStop=/app/nexus/bin/nexus stop
User=nexus
Restart=on-abort
[Install]
WantedBy=multi-user.target
EOT
sudo ln -s /app/nexus/bin/nexus /etc/init.d/nexus
sudo chkconfig --add nexus
sudo chkconfig --levels 345 nexus on
sudo service nexus start
#curl -Ls
https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash
&& sudo NEW_RELIC_API_KEY=${var.nr-key} NEW_RELIC_ACCOUNT_ID=${var.nr-id}
/usr/local/bin/newrelic install -y
sudo hostnamectl set-hostname Nexus
EOF
}
```

Sonarqube server provision
Here we are using sonarqube to scam for code smell,misconfiguration on our application code.all dependencies for our sonarqube server are installed on the userdatascript

In this code, we created a security group resource block (resource "aws_security_group" "sonarqube_sg")for our sonarqube server which is opened on port 22 for only bastion host , other opened port include port 9000 which is sonarqube port opened for ingress to only sonarqube elastic load balancers.
Next is to create the Sonarqube server (resource "aws_instance" "sonarqube-server)on the public subnet together with the userdatascript that helps to install all dependencies the sonarqube server uses, also we added an elastic load balancers resource block (resource "aws_security_group" "elb-sonar-sg") and its security group(resource "aws_security_group" "elb-sonar-sg") . The sonarqube elb security group is opened on port 443 to interface with internet securely, while the load balancer listens on port 9000.Finally we created a record for our sonarqube server using the (rresource "aws_route53_record" "sonar-record") block

```
# Ubuntu AMI lookup
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}
```

```
resource "aws_instance" "sonarqube-server" {
  ami                         = data.aws_ami.ubuntu.id # ubuntu in eu-west-2
  instance_type               = "t2.medium"
  key_name                    = var.key
  subnet_id                   = var.subnet_id
  associate_public_ip_address = true
  vpc_security_group_ids      = [aws_security_group.sonarqube_sg.id]
  root_block_device {
    volume_size = 20    # Size in GB
    volume_type = "gp3" # General Purpose SSD (recommended)
    encrypted   = true  # Enable encryption (best practice)
  }
  user_data = local.userdata
  tags = {
    Name = "${var.name}-sonarqube-server"
  }
}

# Create sonarqube security group
resource "aws_security_group" "sonarqube_sg" {
  name        = "${var.name}-sonarqube-sg"
  description = "Allow SSH and HTTPS"
  vpc_id      = var.vpc-id
  ingress {
    from_port       = 22
    to_port         = 22
    protocol        = "tcp"
    security_groups = [var.bastion]
  }
  ingress {
    from_port   = 9000
    to_port     = 9000
    protocol    = "tcp"
 security_groups = [aws_security_group.elb-sonar-sg.id]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-sonarqube-sg"
  }
}

#Create Security Group for Sonarqube Sever ELB
resource "aws_security_group" "elb-sonar-sg" {
  name        = "${var.name}-sonarqube-elb-sg"
  description = "Allow HTTPS"
  vpc_id      = var.vpc-id

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
```

```
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-sonarqube-elb-sg"
  }
}

# Create Elastic load balancer for Sonarqube Server
resource "aws_elb" "elb-sonar" {
  name            = "${var.name}-elb-sonar"
  subnets         = var.public_subnets
  security_groups = [aws_security_group.elb-sonar-sg.id]

  listener {
    instance_port      = 9000
    instance_protocol  = "http"
    lb_port            = 443
    lb_protocol        = "https"
    ssl_certificate_id = var.acm_certificate_arn
  }

  health_check {
    healthy_threshold   = 2
    unhealthy_threshold = 2
    timeout             = 3
    target              = "TCP:9000"
    interval            = 30
  }

  instances                   = [aws_instance.sonarqube-server.id]
  cross_zone_load_balancing   = true
  idle_timeout                = 400
  connection_draining         = true
  connection_draining_timeout = 400

  tags = {
    Name = "${var.name}-elb-sonar"
  }
}

# Create Route 53 Hosted Zone for Sonarqube
data "aws_route53_zone" "zone_id" {
  name         = var.domain
  private_zone = false
}

# Create Route 53 A Record for Sonarqube Server
resource "aws_route53_record" "sonar-record" {
  zone_id = data.aws_route53_zone.zone_id.zone_id
  name    = "sonar.${var.domain}"
  type    = "A"
```

```
  alias {
    name                   = aws_elb.elb-sonar.dns_name
    zone_id                = aws_elb.elb-sonar.zone_id
    evaluate_target_health = true
  }
}
```

sonarqube userdatascript.

```
locals {
userdata = <<EOF
#!/bin/bash
sudo apt update -y
echo "***Firstly Modify OS Level values***"
sudo bash -c 'echo "
vm.max_map_count=262144
fs.file-max=65536
ulimit -n 65536
ulimit -u 4096" >> /etc/sysctl.conf'
sudo bash -c 'echo "
sonarqube   -   nofile   65536
sonarqube   -   nproc    4096" >> /etc/security/limits.conf'
echo "***********Install Java JDK**********"
sudo apt install openjdk-11-jdk -y
echo "***********Install PostgreSQL**********"
echo "***********The version of postgres currenlty is 14.5 which is not
supported so we have to download v12**********"
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -
cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add -
sudo apt-get update -y
sudo apt-get -y install postgresql-12 postgresql-contrib-12
echo "*****Enable and start, so it starts when system boots up*******"
sudo systemctl enable postgresql
sudo systemctl start postgresql
#Change default password of postgres user
sudo chpasswd <<<"postgres:Admin123@"
#Create user sonar without switching technically
sudo su -c 'createuser sonar' postgres
#Create SonarQube Database and change sonar password
sudo su -c "psql -c \"ALTER USER sonar WITH ENCRYPTED PASSWORD 'Admin123'\""
postgres
sudo su -c "psql -c \"CREATE DATABASE sonarqube OWNER sonar\"" postgres
sudo su -c "psql -c \"GRANT ALL PRIVILEGES ON DATABASE sonarqube to sonar\""
postgres
#Restart postgresql for changes to take effect
sudo systemctl restart postgresql
#Install SonarQube
sudo mkdir /sonarqube/
cd /sonarqube/
sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-
8.6.0.39681.zip
sudo apt install unzip -y
sudo unzip sonarqube-8.6.0.39681.zip -d /opt/
sudo mv /opt/sonarqube-8.6.0.39681/ /opt/sonarqube
```

```
#Add group user sonarqube
sudo groupadd sonar
#Then, create a user and add the user into the group with directory
permission to the /opt/ directory
sudo useradd -c "SonarQube - User" -d /opt/sonarqube/ -g sonar sonar
#Change ownership of the directory to sonar
sudo chown sonar:sonar /opt/sonarqube/ -R
sudo bash -c 'echo "
sonar.jdbc.username=sonar
sonar.jdbc.password=Admin123
sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube
sonar.search.javaOpts=-Xmx512m -Xms512m -XX:+HeapDumpOnOutOfMemoryError"
>> /opt/sonarqube/conf/sonar.properties'
 #Configure such that SonarQube starts on boot up
sudo touch /etc/systemd/system/sonarqube.service
#Configuring so that we can run commands to start, stop and reload sonarqube
service
sudo bash -c 'echo "
[Unit]
Description=SonarQube service
After=syslog.target network.target
[Service]
Type=forking
ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start
ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop
ExecReload=/opt/sonarqube/bin/linux-x86-64/sonar.sh restart
User=sonar
Group=sonar
Restart=always
LimitNOFILE=65536
LimitNPROC=4096
[Install]
WantedBy=multi-user.target" >> /etc/systemd/system/sonarqube.service'
#Enable and Start the Service
sudo systemctl daemon-reload
sudo systemctl enable sonarqube.service
sudo systemctl start sonarqube.service
#Install net-tools incase we want to debug later
sudo apt install net-tools -y
#Install nginx
sudo apt-get install nginx -y
#Configure nginx so we can access server from outside
sudo touch /etc/nginx/sites-enabled/sonarqube.conf
sudo bash -c 'echo "
server {
  listen 80;
  access_log  /var/log/nginx/sonar.access.log;
  error_log   /var/log/nginx/sonar.error.log;
  proxy_buffers 16 64k;
  proxy_buffer_size 128k;
  location / {
      proxy_pass  http://127.0.0.1:9000;
      proxy_next_upstream error timeout invalid_header http_500 http_502
http_503 http_504;
      proxy_redirect off;
      proxy_set_header    Host            \$host;
      proxy_set_header    X-Real-IP       \$remote_addr;
```

```
      proxy_set_header    X-Forwarded-For \$proxy_add_x_forwarded_for;
      proxy_set_header    X-Forwarded-Proto http;
  }
}" >> /etc/nginx/sites-enabled/sonarqube.conf'
#Remove the default configuration file
sudo rm /etc/nginx/sites-enabled/default
#Enable and restart nginix service
sudo systemctl enable nginx.service
sudo systemctl stop nginx.service
sudo systemctl start nginx.service
curl -Ls
https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash
&& sudo NEW_RELIC_API_KEY=${var.nr-key} NEW_RELIC_ACCOUNT_ID=${var.nr-id}
/usr/local/bin/newrelic install -y
sudo hostnamectl set-hostname Sonarqube
sudo reboot
EOF
}
```

database server

Create a Multi az database. here data is persisted on the primary database and later synchronised to the secondary database. Also not that the security group of the database was open to access to only bastion host, prod and stage server on rds port 3306. this is because the stage and prod servers needs to have access to persist data also the bastion host needs have access for testing purposes.

```
#  RDS Subnet Group
resource "aws_db_subnet_group" "db_subnet_group" {
  name        = "${var.name}-db_subnet"
  subnet_ids  = [var.pri-sub-1, var.pri-sub-2] # Use private subnets for
security
  description = "Subnet group for Multi-AZ RDS deployment"

  tags = {
    Name = "${var.name}-db-Subnet-Group"
  }
}
data "vault_generic_secret" "vault-secret" {
  path = "secret/database"
}


resource "aws_db_instance" "mysql_database" {
  identifier             = "${var.name}-db"
  db_subnet_group_name   = aws_db_subnet_group.db_subnet_group.name
  vpc_security_group_ids = [aws_security_group.RDS-sg.id]
  db_name                = "petadoption"
  # High Availability
  multi_az = true

  # Engine Settings
  engine               = "mysql"
  engine_version       = "5.7"
  instance_class       = "db.t3.micro"
```

```
  parameter_group_name = "default.mysql5.7"

  # Storage
  allocated_storage = 20
  storage_type      = "gp3"
  storage_encrypted = true
  # Credentials (Fetch from Vault Manager)
  username = data.vault_generic_secret.vault-secret.data["username"]
  password = data.vault_generic_secret.vault-secret.data["password"]
  # Backup & Maintenance
  skip_final_snapshot = true
  # Security
  publicly_accessible = false
  deletion_protection = false
}
#RDS security group
resource "aws_security_group" "RDS-sg" {
  name        = "${var.name}-rds-sg"
  description = "RDS Security group"
  vpc_id      = var.vpc-id

  ingress {
    description     = "mysqlport"
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [var.bastion, var.stage-sg, var.prod-sg]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.name}-rds-sg"
  }
}
```

Production server
This is is server that houses the contanarized application. here we used the auto-scaling group to deploy this environment this is also to increase availability so as to have a backup whenever we have a faulty production environment.

In the code, we have the security group of the production environment opened up on port 22 to allow access to only our bastion host and ansible server. this is because the ansible needs to have acces to configure the server whenever anew server is spinned up and the bastion host aswell needs access to access the server incase the administrator wants to check for any configuration on the server.
Also security group for the prod application load balancer was also created withaccess to only port 443 to allow https access through the load balancers to the docker host(production server)
A record was also created for the production server.

```
#prod security group
resource "aws_security_group" "prod-sg" {
  name        = "${var.name}-prod-sg"
  description = "prod Security group"
  vpc_id      = var.vpc-id
  ingress {
    description     = "SSH access from bastion"
    from_port       = 22
    to_port         = 22
    protocol        = "tcp"
    security_groups = [var.bastion, var.ansible]
  }

  ingress {
    description = "HTTP access from ALB"
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    description = "Allow all outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-prod-sg"
  }
}
data "aws_ami" "redhat" {
  most_recent = true
  owners      = ["309956199498"] # RedHat's owner ID
  filter {
    name   = "name"
    values = ["RHEL-9*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}
# Create Launch Template
resource "aws_launch_template" "prod_lnch_tmpl" {
  image_id      = data.aws_ami.redhat.id
  name_prefix   = "${var.name}-prod-web-tmpl"
  instance_type = "t2.medium"
  key_name      = var.key-name
  user_data = base64encode(templatefile("./module/prod-env/docker-script.sh",
{
    nexus-ip            = var.nexus-ip,
    nr-key              = var.nr-key,
```

```
    nr-acct-id           = var.nr-acct-id
  }))

  network_interfaces {
    security_groups = [aws_security_group.prod-sg.id]
  }
  #user_data = ""
}

# Create Auto Scaling Group
resource "aws_autoscaling_group" "prod_autoscaling_grp" {
  name                      = "${var.name}-prod-asg"
  max_size                  = 3
  min_size                  = 1
  desired_capacity          = 1
  health_check_grace_period = 120
  health_check_type         = "EC2"
  force_delete              = true
  launch_template {
    id      = aws_launch_template.prod_lnch_tmpl.id
    version = "$Latest"
  }
  vpc_zone_identifier = [var.pri-subnet1, var.pri-subnet2]
  target_group_arns   = [aws_lb_target_group.prod-target-group.arn]

  tag {
    key                 = "Name"
    value               = "${var.name}-prod-asg"
    propagate_at_launch = true
  }
}
# Created autoscaling group policy
resource "aws_autoscaling_policy" "prod-asg-policy" {
  name                   = "asg-policy"
  adjustment_type        = "ChangeInCapacity"
  autoscaling_group_name = aws_autoscaling_group.prod_autoscaling_grp.name
  policy_type            = "TargetTrackingScaling"
  target_tracking_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ASGAverageCPUUtilization"
    }
    target_value = 70.0
  }
}

# Create Application Load Balancer for prod
resource "aws_lb" "prod_LB" {
  name                       = "${var.name}-prod-LB"
  internal                   = false
  load_balancer_type         = "application"
  security_groups            = [aws_security_group.prod-sg.id]
  subnets                    = [var.pub-subnet1, var.pub-subnet2]
  tags = {
    Name = "${var.name}-prod-LB"
  }
}
#prod-elb security group
```

```
resource "aws_security_group" "prod-elb-sg" {
  name        = "${var.name}-prod-elb-sg"
  description = "prod-elb Security group"
  vpc_id      = var.vpc-id
  ingress {
    description = "HTTP access from ALB"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    description = "Allow all outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-prod-elb-sg"
  }
}

#Create Target group for load Balancer
resource "aws_lb_target_group" "prod-target-group" {
  name        = "${var.name}-prod-tg"
  port        = 8080
  protocol    = "HTTP"
  vpc_id      = var.vpc-id
  target_type = "instance"
  health_check {
    healthy_threshold   = 3
    unhealthy_threshold = 5
    interval            = 30
    timeout             = 5
    path                = "/"
  }
  tags = {
    Name = "${var.name}-prod-tg"
  }
}

# Create load balance listener for http
resource "aws_lb_listener" "prod_load_balancer_listener_http" {
  load_balancer_arn = aws_lb.prod_LB.arn
  port              = "80"
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.prod-target-group.arn
  }
}
# Create load balance listener for https
resource "aws_lb_listener" "prod_load_balancer_listener_https" {
  load_balancer_arn = aws_lb.prod_LB.arn
  port              = "443"
  protocol          = "HTTPS"
```

```
  ssl_policy        = "ELBSecurityPolicy-2016-08"
  certificate_arn   = var.acm-cert-arn
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.prod-target-group.arn
  }
}
# Create Route 53 record for prod server
data "aws_route53_zone" "team2-acp-zone" {
  name         = var.domain
  private_zone = false
}

# Create Route 53 record for prod server
resource "aws_route53_record" "prod-record" {
  zone_id = data.aws_route53_zone.team2-acp-zone.zone_id
  name    = "www.${var.domain}"
  type    = "A"
  alias {
    name                   = aws_lb.prod_LB.dns_name
    zone_id                = aws_lb.prod_LB.zone_id
    evaluate_target_health = true
  }
}
```

prod server script

The major function for this production script is to install docker in the server and make it ready to receive docker image for our application deployment. It has in the script a file called script.sh which can only be executed when we want to pull the latest docker image from our nexus docker registry. the function of this script.sh is to login to the nexus docker registry,pull the latest image of our application stop and remove already running container, deploy the new image and the container on the server.

```
#!/bin/bash

#updatng the instance
sudo yum update -y
sudo yum upgrade -y

#install Docker and its dependencies, start Docker service
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo #downloads & adds
repo to instance
sudo yum install docker-ce -y



#add a registry to the Docker daemon configuration to allow
#insecure communication (without TLS verification) with a Docker registry on
port 8085
sudo cat <<EOT>> /etc/docker/daemon.json
  {
    "insecure-registries" : ["${nexus-ip}:8085"]
  }
```

```
EOT

#Starts the Docker service and enables it to run on boot.
#Add the ec2-user to the docker group, allowing them to run Docker commands.
sudo service docker start
sudo systemctl start docker
sudo systemctl enable docker

#add the ec2-user to the docker group, allowing them to run Docker commands.
sudo usermod -aG docker ec2-user

#create a shell script that manages a Docker container on the EC2 instance,
pulling updates from a Nexus registry
#i.e. Configure docker to fetch image from Nexus
sudo mkdir /home/ec2-user/scripts
cat << EOF > "/home/ec2-user/scripts/script.sh"
#!/bin/bash

set -x

#Define Variables
IMAGE_NAME="${nexus-ip}:8085/petclinicapps"
CONTAINER_NAME="appContainer"
NEXUS_IP="${nexus-ip}:8085"

#Function to Login to dockerhub
authenticate_docker() {
    docker login --username=admin --password=admin123 \$NEXUS_IP
}

#Function to check for latest image on dockerhub
check_for_updates() {
    local latest_image=\$(docker pull \$IMAGE_NAME | grep "Status: Image is
up to date" | wc -l)
    if [ \$latest_image -eq 0 ]; then
        return 0
    else
        return 1
    fi
}

#Function to stop and remove the current container
#Function to deploy image in a container
update_container() {
  docker stop \$CONTAINER_NAME
  docker rm \$CONTAINER_NAME
  docker run -d --name \$CONTAINER_NAME -p 8080:8080 \$IMAGE_NAME
}

#Main Function
main() {
    authenticate_docker
    if check_for_updates; then
        update_container
        echo "Container upgraded to latest image."
    else
        echo "Up to date! No image update required. Exiting..."
```

```
    fi
}
main
EOF

sudo chown -R ec2-user:ec2-user /home/ec2-user/scripts/script.sh
sudo chmod 777 /home/ec2-user/scripts/script.sh

#Restart Docker
sudo systemctl restart docker

#Install New Relic CLI, Set hostname for the instance
curl -Ls
https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash
&& sudo NEW_RELIC_API_KEY="${nr-key}" NEW_RELIC_ACCOUNT_ID="${nr-acct-id}"
NEW_RELIC_REGION="EU" /usr/local/bin/newrelic install -y

#setting the host name
sudo hostnamectl set-hostname prod-instance
sudo reboot
```

Stage server
This is is server that houses the contanarized application. here we used the auto-scaling group to deploy this environment this is also to increase availability so as to have a backup whenever we have a faulty production environment.

In the code, we have the security group of the stage environment opened up on port 22 to allow access to only our bastion host and ansible server. this is because the ansible needs to have acces to configure the server whenever anew server is spinned up and the bastion host aswell needs access to access the server incase the administrator wants to check for any configuration on the server.
Also security group for the prod application load balancer was also created withaccess to only port 443 to allow https access through the load balancers to the docker host(stage server)
A record was also created for the production server.

```
#stage security group
resource "aws_security_group" "stage-sg" {
  name        = "${var.name}-stage-sg"
  description = "stage Security group"
  vpc_id      = var.vpc-id
  ingress {
    description     = "SSH access from bastion"
    from_port       = 22
    to_port         = 22
    protocol        = "tcp"
    security_groups = [var.bastion, var.ansible]
  }

  ingress {
    description = "HTTP access from ALB"
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
```

```
    cidr_blocks = ["0.0.0.0/0"]
    # security_groups = [aws_security_group.stage-elb-sg.id]
  }
  egress {
    description = "Allow all outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-stage-sg"
  }
}
data "aws_ami" "redhat" {
  most_recent = true
  owners      = ["309956199498"] # RedHat's owner ID
  filter {
    name   = "name"
    values = ["RHEL-9*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}
# Create Launch Template
resource "aws_launch_template" "stage_lnch_tmpl" {
  image_id      = data.aws_ami.redhat.id
  name_prefix   = "${var.name}-stage-web-tmpl"
  instance_type = "t2.medium"
  key_name      = var.key-name
  user_data = base64encode(templatefile("./module/stage-env/docker-
script.sh", {
    nexus-ip            = var.nexus-ip,
    nr-key              = var.nr-key,
    nr-acct-id          = var.nr-acct-id
  }))

  network_interfaces {
    security_groups = [aws_security_group.stage-sg.id]
  }
  #user_data = ""
}

# Create Auto Scaling Group
resource "aws_autoscaling_group" "stage_autoscaling_grp" {
  name                      = "${var.name}-stage-asg"
  max_size                  = 3
  min_size                  = 1
  desired_capacity          = 1
  health_check_grace_period = 120
  health_check_type         = "EC2"
```

```
  force_delete              = true
  launch_template {
    id      = aws_launch_template.stage_lnch_tmpl.id
    version = "$Latest"
  }
  vpc_zone_identifier = [var.pri-subnet1, var.pri-subnet2]
  target_group_arns   = [aws_lb_target_group.stage-target-group.arn]

  tag {
    key                 = "Name"
    value               = "${var.name}-stage-asg"
    propagate_at_launch = true
  }
}
# Created autoscaling group policy
resource "aws_autoscaling_policy" "stage-asg-policy" {
  name                   = "asg-policy"
  adjustment_type        = "ChangeInCapacity"
  autoscaling_group_name = aws_autoscaling_group.stage_autoscaling_grp.name
  policy_type            = "TargetTrackingScaling"
  target_tracking_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ASGAverageCPUUtilization"
    }
    target_value = 70.0
  }
}

# Create Application Load Balancer for stage
resource "aws_lb" "stage_LB" {
  name                       = "${var.name}-stage-LB"
  internal                   = false
  load_balancer_type         = "application"
  security_groups            = [aws_security_group.stage-elb-sg.id]
  subnets                    = [var.pub-subnet1, var.pub-subnet2]
  tags = {
    Name = "${var.name}-stage-LB"
  }
}
#stage-elb security group
resource "aws_security_group" "stage-elb-sg" {
  name        = "${var.name}-stage-elb-sg"
  description = "stage-elb Security group"
  vpc_id      = var.vpc-id
  ingress {
    description = "HTTP access from ALB"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    description = "Allow all outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
```

```
  }
  tags = {
    Name = "${var.name}-stage-elb-sg"
  }
}

#Create Target group for load Balancer
resource "aws_lb_target_group" "stage-target-group" {
  name        = "${var.name}-stage-tg"
  port        = 8080
  protocol    = "HTTP"
  vpc_id      = var.vpc-id
  target_type = "instance"
  health_check {
    healthy_threshold   = 3
    unhealthy_threshold = 5
    interval            = 30
    timeout             = 5
    path                = "/"
  }
  tags = {
    Name = "${var.name}-stage-tg"
  }
}

# Create load balance listener for http
resource "aws_lb_listener" "stage_load_balancer_listener_http" {
  load_balancer_arn = aws_lb.stage_LB.arn
  port              = "80"
  protocol          = "HTTP"
   default_action {
    type = "redirect"

    redirect {
      port        = 443
      protocol    = "HTTPS"
      status_code = "HTTP_301"
    }
  }
}
# Create load balance listener for https
resource "aws_lb_listener" "stage_load_balancer_listener_https" {
  load_balancer_arn = aws_lb.stage_LB.arn
  port              = "443"
  protocol          = "HTTPS"
  ssl_policy        = "ELBSecurityPolicy-2016-08"
  certificate_arn   = var.acm-cert-arn
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.stage-target-group.arn
  }
}
# Create Route 53 record for stage server
data "aws_route53_zone" "team2-acp-zone" {
  name         = var.domain
  private_zone = false
}
```

```
# Create Route 53 record for stage server
resource "aws_route53_record" "stage-record" {
  zone_id = data.aws_route53_zone.team2-acp-zone.zone_id
  name    = "stage.${var.domain}"
  type    = "A"
  alias {
    name                   = aws_lb.stage_LB.dns_name
    zone_id                = aws_lb.stage_LB.zone_id
    evaluate_target_health = true
  }
}
```

stage server script

The major function for this production script is to install docker in the server and make it ready to receive docker image for our application deployment. It has in the script a file called script.sh which can only be executed when we want to pull the latest docker image from our nexus docker registry. the function of this script.sh is to login to the nexus docker registry,pull the latest image of our application stop and remove already running container, deploy the new image and the container on the server.

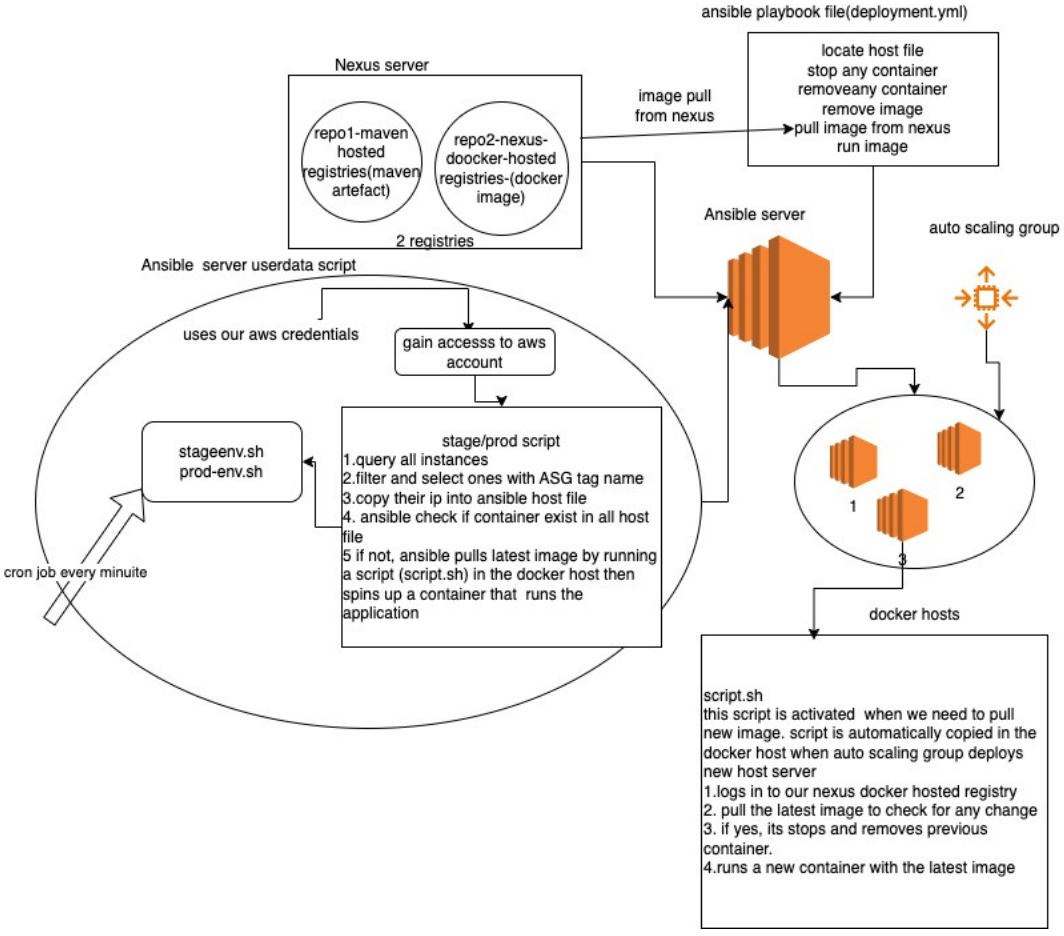
```
#!/bin/bash

#updatng the instance
sudo yum update -y
sudo yum upgrade -y

#install Docker and its dependencies, start Docker service
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo #downloads & adds
repo to instance
sudo yum install docker-ce -y



#add a registry to the Docker daemon configuration to allow
#insecure communication (without TLS verification) with a Docker registry on
port 8085
sudo cat <<EOT>> /etc/docker/daemon.json
  {
    "insecure-registries" : ["${nexus-ip}:8085"]
  }
EOT

#Starts the Docker service and enables it to run on boot.
#Add the ec2-user to the docker group, allowing them to run Docker commands.
sudo service docker start
sudo systemctl start docker
sudo systemctl enable docker

#add the ec2-user to the docker group, allowing them to run Docker commands.
sudo usermod -aG docker ec2-user
```

```
#create a shell script that manages a Docker container on the EC2 instance,
pulling updates from a Nexus registry
#i.e. Configure docker to fetch image from Nexus
sudo mkdir /home/ec2-user/scripts
cat << EOF > "/home/ec2-user/scripts/script.sh"
#!/bin/bash

set -x

#Define Variables
IMAGE_NAME="${nexus-ip}:8085/petclinicapps"
CONTAINER_NAME="appContainer"
NEXUS_IP="${nexus-ip}:8085"

#Function to Login to dockerhub
authenticate_docker() {
    docker login --username=admin --password=admin123 \$NEXUS_IP
}

#Function to check for latest image on dockerhub
check_for_updates() {
    local latest_image=\$(docker pull \$IMAGE_NAME | grep "Status: Image is
up to date" | wc -l)
    if [ \$latest_image -eq 0 ]; then
        return 0
    else
        return 1
    fi
}

#Function to stop and remove the current container
#Function to deploy image in a container
update_container() {
  docker stop \$CONTAINER_NAME
  docker rm \$CONTAINER_NAME
  docker run -d --name \$CONTAINER_NAME -p 8080:8080 \$IMAGE_NAME
}

#Main Function
main() {
    authenticate_docker
    if check_for_updates; then
        update_container
        echo "Container upgraded to latest image."
    else
        echo "Up to date! No image update required. Exiting..."
    fi
}
main
EOF

sudo chown -R ec2-user:ec2-user /home/ec2-user/scripts/script.sh
sudo chmod 777 /home/ec2-user/scripts/script.sh

#Restart Docker
sudo systemctl restart docker
```

```
#Install New Relic CLI, Set hostname for the instance
curl -Ls
https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash
&& sudo NEW_RELIC_API_KEY="${nr-key}" NEW_RELIC_ACCOUNT_ID="${nr-acct-id}"
NEW_RELIC_REGION="EU" /usr/local/bin/newrelic install -y

#setting the host name
sudo hostnamectl set-hostname stage-instance
sudo reboot
```

E. ANSIBLE CODE AND WORKFLOW EXPLANATION

Ansible server workflow



Here, the ansible server generally helps us to configure the docker host server and deploy the

application with the help of the ansible playbook.
Our Jenkinsfile houses various stages in our application pipeline, of which one of the stages is to deploy the application in our production/stage environment. When that stage is triggered, Our ansible server is triggered to run the ansible playbook.This is what happens when the ansible playbook is triggered:

It starts with the Ansible locating the host file where it would run the docker image of our application, It then stops and removes any running container together with the previous image if any.

Thereafter it finally pulls the latest image from the nexus docker hosted repository and runs the application.
NB : this process happens when ever there is a change in the code in our application pipeline.
Here comes the big question ???? how does Ansible update its host file with the latest ip addresses of the docker host (servers) based on the fact that all our docker host are deployed using autoscaling group that scales in when traffic reduces and scales out when traffic increases, how does Ansible learn about the new deployed servers on increased traffic?
This is how the ips of the docker servers are updated on the host file.
Ansible server is deployed along side an ansible userdata script. In the userdata script, we have set a cron job that runs every minute to execute our production/stage bashscript script every minute.
The production or stage script function is to query all the ec2 instances in our aws account and filter all ec2 instances created by our autoscaling group using the auto scaling group tag name with the help of our aws credentials already provided for it.

After a successful query is done, it copies these filtered ipaddress and replaces the host file of the ansible with the latest ip addresses it obtained after filtering.
In same script, ansible logins to each and every ip to check if there are any container running the application already, if yes, it skips that docker host to the next one and if it finds host with no latest container, it pulls the latest docker image using another script called (script.sh) that is stored already in the docker host when it was deployed by the auto scaling group.

The functon of this “script.sh“ is to login to our nexus docker registry to pull the latest version of the specified docker image in that repository, if newer version exist, downloads it, it stops and removes any old running container and redeploys the new version image in a new container. But if the output from the latest image version pulled by the docker has a status of”Image is up to date” that means that the old image is still the latest image and it does nothing and exits.
NB:Note that all these activity on our production/stage script runs every minute with the help of cron job that we set on our Ansible script.
so with this in place our Ansible host file is well updated every minute to get all host does some check to deploy our latest application image on our host servers if they do not exist.

Ansible main.tf file

ansible server was deployed using a redhat instance.We created a folder in our ansible module which we called “scripts“ that houses our ansible playbook(deployment.yml), prduction and stage scripts. with the help of a null resource creation we were able to copy the above mentioned files to our s3 bucket created initially.

```
resource "null_resource" "copy_ansible_playbooks" {
  provisioner "local-exec" {
    command = <<EOT
      aws s3 cp --recursive ${path.module}/scripts/
s3://${var.s3Bucket}/ansible/
    EOT
  }
  depends_on = [time_sleep.wait_for_ansible]
}
```

Ansible security group was also created so as to allow ssh from the bastion host only, using (resource "aws_security_group" "ansible-sg")
IAM rols was also created and s3full access policy was attached to the role to enable ansible server have full access to perform operations on s3 bucket
Also IAm user was created to enable ansible perform operations /login to other ec2 servers to perform operation

```
# Data source to get the latest RedHat AMI
data "aws_ami" "redhat" {
  most_recent = true
  owners      = ["309956199498"] # RedHat's owner ID
  filter {
    name   = "name"
    values = ["RHEL-9*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}
# create a time sleep resource to wait for the ansible server to be up and
running
resource "time_sleep" "wait_for_ansible" {
  depends_on = [aws_instance.ansible-server]
  create_duration = "15s"
}
# null resurce to copy the ansible playbooks folder into the s3 bucket
resource "null_resource" "copy_ansible_playbooks" {
  provisioner "local-exec" {
    command = <<EOT
      aws s3 cp --recursive ${path.module}/scripts/
s3://${var.s3Bucket}/ansible/
    EOT
  }
  depends_on = [time_sleep.wait_for_ansible]
```

```
}

# Create Ansible Server
resource "aws_instance" "ansible-server" {
  ami                    = data.aws_ami.redhat.id #rehat
  instance_type          = "t2.medium"
  iam_instance_profile= aws_iam_instance_profile.s3-bucket-instance-
profile.name
  vpc_security_group_ids = [aws_security_group.ansible-sg.id]
  key_name               = var.keypair
  subnet_id              = var.subnet_id
  user_data              = local.ansible_userdata
  root_block_device {
    volume_size = 20
    volume_type = "gp3"
    encrypted   = true
  }
  tags = {
    Name = "${var.name}-ansible-server"
  }
}

#Creating ansible secority group
resource "aws_security_group" "ansible-sg" {
  name        = "${var.name}ansible-sg"
  description = "Allow ssh"
  vpc_id      = var.vpc

  ingress {
    description     = "sshport"
    from_port       = 22
    to_port         = 22
    protocol        = "tcp"
    security_groups = [var.bastion]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "${var.name}-ansible-sg"
  }
}

# IAM User
resource "aws_iam_user" "ansible-user" {
  name = "${var.name}-ansible-user"
}

resource "aws_iam_group" "ansible-group" {
  name = "${var.name}-ansible-group"
}

resource "aws_iam_access_key" "ansible-user-key" {
```

```
  user = aws_iam_user.ansible-user.name
}

resource "aws_iam_user_group_membership" "ansible-group-member" {
  user   = aws_iam_user.ansible-user.name
  groups = [aws_iam_group.ansible-group.name]
}

resource "aws_iam_group_policy_attachment" "ansible-policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
  group      = aws_iam_group.ansible-group.name
}



# Create IAM role for ansible server
resource "aws_iam_role" "s3-bucket-role" {
  name = "${var.name}-ansible-bucket-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          Service = "ec2.amazonaws.com"
        },
        Action = "sts:AssumeRole"
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "ansible-bucket-role-attachment" {
  role       = aws_iam_role.s3-bucket-role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess"
}
# Attach the role to the instance
resource "aws_iam_instance_profile" "s3-bucket-instance-profile" {
  name = "${var.name}-s3-bucket-instance-profile"
  role = aws_iam_role.s3-bucket-role.name
}
```

Production bashscript

This production environment script contains functions that queries all Ec2 instances in our AWS account to fish out all instances with the autoscaling group tag name as well as the instance ip addresses.

it fetches the ip, pushes the ip’s into the Ansible inventory file, logs into them one by one to check if they are running the latest container image and if not, it executes a preinstalled script(script.sh) that pull the lates image from the nexus docker repository to run the application.

```
#!/bin/bash
set -x
```

```
#defining our variables
AWSCLI_PATH='/usr/local/bin/aws'
INVENTORY_FILE='/etc/ansible/prod_hosts'
IPS_FILE='/etc/ansible/prod.lists'
ASG_NAME='team2-ad-prod-asg'
SSH_KEY_PATH='~/.ssh/id_rsa'
WAIT_TIME=20

#write our functions
#fetching the IPs
find_ips() {
    \$AWSCLI_PATH ec2 describe-instances \\
    --filters "Name=tag:aws:autoscaling:groupName,Values=\$ASG_NAME" \\
    --query
'Reservations[*].Instances[*].NetworkInterfaces[*].PrivateIpAddress' \\
    --output text > "\$IPS_FILE"
}
#update the inventory files
update_inventory() {
    echo "[webservers]" > "\$INVENTORY_FILE"
    while IFS= read -r instance; do
        ssh-keyscan -H "\$instance" >> ~/.ssh/known_hosts
        echo "\$instance ansible_user=ec2-user" >> "\$INVENTORY_FILE"
    done < "\$IPS_FILE"
    echo "Inventory updated succesfully"
}
#wait for some minutes
wait_for_seconds() {
    echo "Waiting for \$WAIT_TIME seconds..."
    sleep "\$WAIT_TIME"
}
# check if the docker container is running on all the instances in the ips
file by sshing into them one by one,
# if container is not running on the instance, then execute a script
/home/ec2-user/scripts.sh to start the container
check_docker_container() {
    while IFS= read -r instance; do
        ssh -i "\$SSH_KEY_PATH" ec2-user@"\$instance" "docker ps | grep
appContainer" > /dev/null 2>&1
        if [ $? -ne 0 ]; then
            echo "Container not running on \$instance. Starting container..."
            ssh -i "\$SSH_KEY_PATH" ec2-user@"\$instance" "bash /home/ec2-
user/scripts/script.sh"
        else
            echo "Container is running on \$instance."
        fi
    done < "\$IPS_FILE"
}
# Main function block
main() {
    find_ips
    update_inventory
    wait_for_seconds
    check_docker_container
}
# Execute main function
main
```

```
### End of script
```

Stage bash script
This Stage environment script contains functions that queries all Ec2 instances in our AWS account to fish out all instances with the autoscaling group tag name as well as the instance ip addresses.
it fetches the ip, pushes the ip's into the Ansible inventory file, logs into them one by one to check if they are running the latest container image and if not, it executes a preinstalled script(script.sh) that pull the lates image from the nexus docker repository to run the application.

```
#!/bin/bash
set -x

# Set up logging
LOG_FILE="/var/log/stage-bashscript.log"
exec > >(tee -a "$LOG_FILE") 2>&1
echo "Stage script started at $(date)"



# Defining our variables
AWSCLI_PATH='/usr/local/bin/aws'
INVENTORY_FILE='/etc/ansible/stage_hosts'
IPS_FILE='/etc/ansible/stage.lists'
ASG_NAME='auto-discov-stage-asg'
SSH_KEY_PATH='/home/ec2-user/.ssh/id_rsa'  # Fixed: Using full path
WAIT_TIME=20

# Ensure directories exist
mkdir -p /etc/ansible
mkdir -p /home/ec2-user/.ssh
chown ec2-user:ec2-user /home/ec2-user/.ssh

# Write our functions

# Fetching the IPs
find_ips() {
    if ! $AWSCLI_PATH ec2 describe-instances \
    --filters "Name=tag:aws:autoscaling:groupName,Values=$ASG_NAME" \
    --query
'Reservations[*].Instances[*].NetworkInterfaces[*].PrivateIpAddress' \
    --output text > "$IPS_FILE"; then
        echo "ERROR: Failed to fetch stage IPs" | tee -a "$LOG_FILE"
        return 1
    fi
    echo "Successfully fetched stage IPs"
}

# Update the inventory files
update_inventory() {
```

```
    echo "[webservers]" > "$INVENTORY_FILE"
    while IFS= read -r instance; do
        if [ -z "$instance" ]; then
            continue
        fi
        ssh-keyscan -H "$instance" >> /home/ec2-user/.ssh/known_hosts
2>/dev/null
        echo "$instance ansible_user=ec2-user
ansible_ssh_private_key_file=$SSH_KEY_PATH" >> "$INVENTORY_FILE"
    done < "$IPS_FILE"
    echo "Stage inventory updated successfully"
}

# Wait for some minutes
wait_for_seconds() {
    echo "Waiting for $WAIT_TIME seconds..."
    sleep "$WAIT_TIME"
}

# Check docker container status
check_docker_container() {
    while IFS= read -r instance; do
        if [ -z "$instance" ]; then
            continue
        fi

        if ! ssh -o StrictHostKeyChecking=no -i "$SSH_KEY_PATH" ec2-
user@"$instance" "docker ps | grep stageContainer" > /dev/null 2>&1; then
            echo "Stage container not running on $instance. Starting
container..." | tee -a "$LOG_FILE"
            if ! ssh -o StrictHostKeyChecking=no -i "$SSH_KEY_PATH" ec2-
user@"$instance" "bash /home/ec2-user/stage-scripts/script.sh"; then
                echo "ERROR: Failed to start stage container on $instance" |
tee -a "$LOG_FILE"
            fi
        else
            echo "Stage container is running on $instance." | tee -a
"$LOG_FILE"
        fi
    done < "$IPS_FILE"
}

# Main function block
main() {
    echo "Starting stage environment maintenance..."

    if ! find_ips; then
        exit 1
    fi

    if ! update_inventory; then
        exit 1
    fi

    wait_for_seconds

    if ! check_docker_container; then
```

```
        exit 1
    fi

    echo "Stage maintenance completed successfully at $(date)"
}

# Execute main function
main

### End of script
```

ansible playbook(deployment.yml file)
the ansible play book starts with identifying the hosts where the application would be deployed and variable file that containes information on docker registry.
this is followed by

```
---
 - hosts: webservers
   become: true
   vars_files:
    - ansible_vars_file.yml
   tasks:
   - name: Stop any container running
     command: docker stop appContainer
     ignore_errors: yes
   - name: Remove stopped container
     command: docker rm appContainer
     ignore_errors: yes
   - name: Remove docker image
     command: docker rmi "{{ NEXUS_IP }}"/petclinicapps:latest
     ignore_errors: yes
   - name: Pull docker image from nexus and create a container
     shell: |
       sudo su -c "docker login --username=admin --password=admin123
{{ NEXUS_IP }}" ec2-user
       sudo su -c "docker pull "{{ NEXUS_IP }}"/petclinicapps:latest" ec2-
user
       sudo su -c "docker run -it -d --name appContainer -p 8080:8080
"{{ NEXUS_IP }}"/petclinicapps:latest" ec2-user
```

root main.tf
In this root main.tf file,we called out all the modules we already created so that terraform would utilise it to provision our infrastructure. here we have called out the bastion, vpc, sonarqube, nexus,ansible, database, production and stage environment modules

```
locals {
  name = "team2-ad"
}

module "bastion" {
  source  = "./module/baston"
  name    = local.name
  vpc     = module.vpc.vpc_id
  keypair = module.vpc.public_key
```

```
  privatekey = module.vpc.private_key
  subnets = [module.vpc.pub_sub1_id, module.vpc.pub_sub2_id]
}
module "vpc" {
  source = "./module/vpc"
  name   = local.name
  az1    = "eu-west-2a"
  az2    = "eu-west-2b"
}
module "sonarqube" {
  source              = "./module/sonarqube"
  key                 = module.vpc.public_key
  name                = local.name
  subnet_id           = module.vpc.pub_sub1_id
  bastion             = module.bastion.bastion-sg
  vpc-id              = module.vpc.vpc_id
  domain              = var.domain
  public_subnets      = [module.vpc.pub_sub1_id, module.vpc.pub_sub2_id]
  acm_certificate_arn = data.aws_acm_certificate.team2-cert.arn
  nr-key              = var.nr-key
  nr-id               = var.nr-id
}
module "nexus" {
  source              = "./module/nexus"
  subnet-id           = module.vpc.pub_sub1_id
  keypair             = module.vpc.public_key
  name                = local.name
  vpc                 = module.vpc.vpc_id
  baston-sg           = module.bastion.bastion-sg
  subnet1_id          = module.vpc.pub_sub1_id
  subnet2_id          = module.vpc.pri_sub2_id
  acm_certificate_arn = data.aws_acm_certificate.team2-cert.arn
  domain              = var.domain
  nr-key              = var.nr-key
  nr-id               = var.nr-id
}
module "ansible" {
  source    = "./module/ansible"
  name      = local.name
  keypair   = module.vpc.public_key
  subnet_id = module.vpc.pri_sub1_id
  vpc       = module.vpc.vpc_id
  bastion   = module.bastion.bastion-sg
  private-key = module.vpc.private_key
  deployment = "./module/ansible/deployment.yml" # Path to the deployment
file
  prod-bashscript = "./module/ansible/prod-bashscript.sh" # Path to the prod
bash script
  stage-bashscript = "./module/ansible/stage-bashscript.sh" # Path to the
stage bash script
  nexus-ip = module.nexus.nexus_ip
  nr-key = var.nr-key
  nr-acc-id = var.nr-id
}
module "database" {
  source    = "./module/database"
  name      = local.name
```

```
  pri-sub-1 = module.vpc.pri_sub1_id
  pri-sub-2 = module.vpc.pri_sub2_id
  bastion   = module.bastion.bastion-sg
  vpc-id    = module.vpc.vpc_id
  stage-sg  = module.stage-env.stage-sg
  prod-sg   = module.prod-env.prod-sg
}
module "prod-env" {
  source       = "./module/prod-env"
  name         = local.name
  vpc-id       = module.vpc.vpc_id
  bastion      = module.bastion.bastion-sg
  key-name     = module.vpc.public_key
  pri-subnet1  = module.vpc.pri_sub1_id
  pri-subnet2  = module.vpc.pri_sub2_id
  pub-subnet1  = module.vpc.pub_sub1_id
  pub-subnet2  = module.vpc.pub_sub2_id
  acm-cert-arn = data.aws_acm_certificate.team2-cert.arn
  domain       = var.domain
  nexus-ip     = module.nexus.nexus_ip
  nr-key       = var.nr-key
  nr-acct-id   = var.nr-id
  ansible =  module.ansible.ansible_sg
}
module "stage-env" {
  source       = "./module/stage-env"
  name         = local.name
  vpc-id       = module.vpc.vpc_id
  bastion      = module.bastion.bastion-sg
  key-name     = module.vpc.public_key
  pri-subnet1  = module.vpc.pri_sub1_id
  pri-subnet2  = module.vpc.pri_sub2_id
  pub-subnet1  = module.vpc.pub_sub1_id
  pub-subnet2  = module.vpc.pub_sub2_id
  acm-cert-arn = data.aws_acm_certificate.team2-cert.arn
  domain       = var.domain
  nexus-ip     = module.nexus.nexus_ip
  nr-key       = var.nr-key
  nr-acct-id   = var.nr-id
  ansible =  module.ansible.ansible_sg
}
data "aws_acm_certificate" "team2-cert" {
  domain   = var.domain
  statuses = ["ISSUED"]
}
```

root provider.tf
This is the document that tells terraform what cloud provider we are using, the profile as well as the region of deployment.
it also provides information on the provider for our vault server as well as the token to be used to log into the vault server.
This is also where we configured our terraform to use the created s3 bucket as a backend to store our terraform state file.

```
provider "aws" {
```

```
  region  = "eu-west-2"
  # profile = "pet-adoption"

}
provider "vault" {
  address = "https://vault.bolatitoadegoroye.top"
  token   = "s.654dBIbmq4zPtdgeyysk3VpM"
}

terraform {
  backend "s3" {
    bucket         = "team2-bucket-pet-adoption"
    key            = "infrastructure/terraform.tfstate"
    region         = "eu-west-2"
    use_lockfile   = true
    encrypt        = true
  }
}
```

F. INFRASTRUCTURE JENKINSFILE EXPLANATION

the jenkins pipelines starts with setting up agent where the pipeline would run, so here we set up any agent which refers to any available agent which is our jenkins server.
next we setup tools that would be used by the pipeline. her we setup terraform as a tool that would be used to provision the infrastructure.
Next we set up trigger which we set up to run every midnight. also we set up environment variable like the SLACKCHANNEL where the team would receive notification about the success or failure of any build, and SLACKCREDENTIALS , wherewe set the crdentials that would be used for slack notification
After that, here comes the stages of the infrastructure pipeline.
It starts with infrastructure scan with checkov , then followed by terraform commnd to provision infrastructure.afterwhich the notification on slack for successful or unsuccessful build on slack.

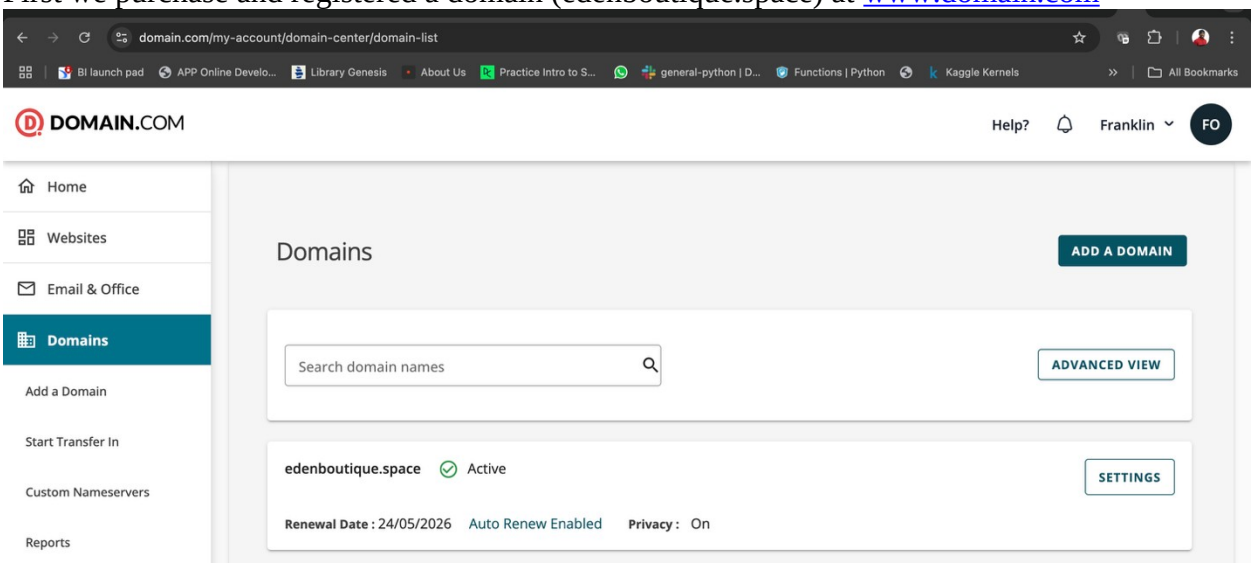
```
pipeline {
    agent any
    tools {
        terraform 'terraform'
    }
    parameters {
        choice(name: 'action', choices: ['apply', 'destroy'], description:
'Select the action to perform')
    }
    triggers {
        cron('H 0 * * *') // Runs every day at midnight
    }
    environment {
        SLACKCHANNEL = 'D08B6QQ2T50' //MY CHANNEL ID
        SLACKCREDENTIALS = credentials('slack')
    }

    stages {
        stage('IAC Scan') {
```

```
            steps {
                script {
                    sh 'pip install pipenv'
                    sh 'pipenv run pip install checkov'
                    def checkovStatus = sh(script: 'pipenv run checkov -d . -
o cli --output-file checkov-results.txt --quiet', returnStatus: true)
                    junit allowEmptyResults: true, testResults: 'checkov-
results.txt'
                    // if (checkovStatus != 0) {
                    //     error 'Checkov found some issues'
                    // }
                }
            }
        }
        stage('Terraform Init') {  // Fixed spelling
            steps {
                sh 'terraform init'
            }
        }
        stage('Terraform format') {
            steps {
                sh 'terraform fmt --recursive'
            }
        }
        stage('Terraform validate') {
            steps {
                sh 'terraform validate'
            }
        }
        stage('Terraform plan') {
            steps {
                sh 'terraform plan'
            }
        }
        stage('Terraform action') {
            steps {
                script {
                    sh "terraform ${action} -auto-approve"
                }
            }
        }
    }
    post {
        always {
            script {
                slackSend(
                    channel: SLACKCHANNEL,
                    color: currentBuild.result == 'SUCCESS' ? 'good' :
'danger',
                    message: "Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' ($
{env.BUILD_URL}) has been completed."
                )
            }
        }
        failure {
            slackSend(
                channel: SLACKCHANNEL,
```

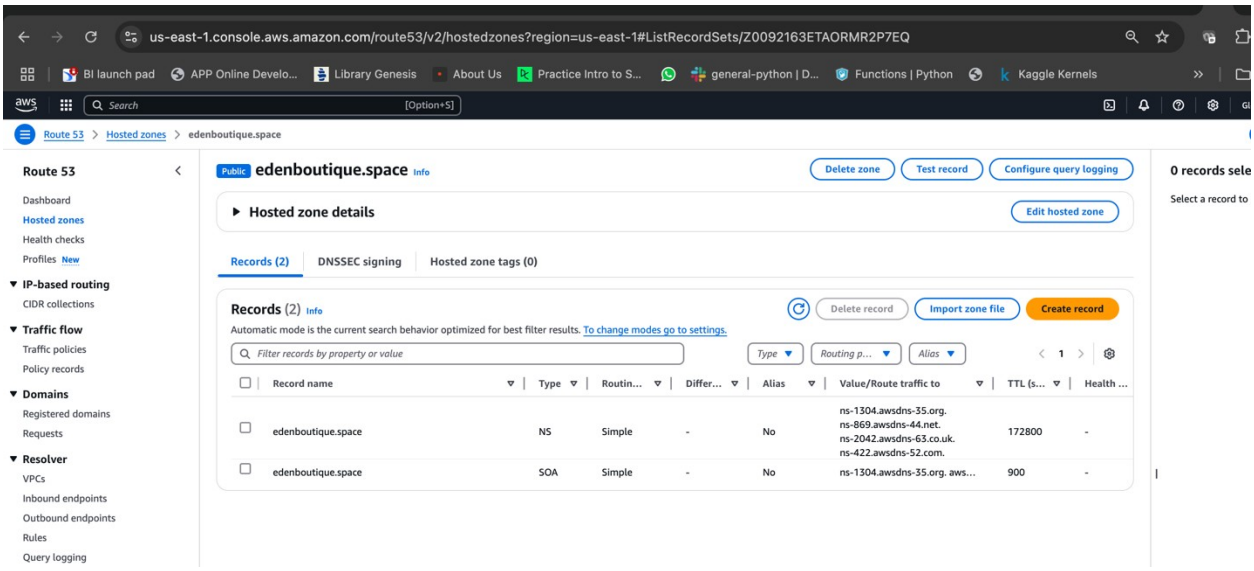
```
                color: 'danger',
                message: "Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' has
failed. Check console output at ${env.BUILD_URL}."
            )
        }
        success {
            slackSend(
                channel: SLACKCHANNEL,
                color: 'good',
                message: "Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'
completed successfully. Check console output at ${env.BUILD_URL}."
            )
        }
    }
}
```

G. INFRASTRUCTURE PROVISIONING SETUP FOR S3,VAULT AND JENKINS

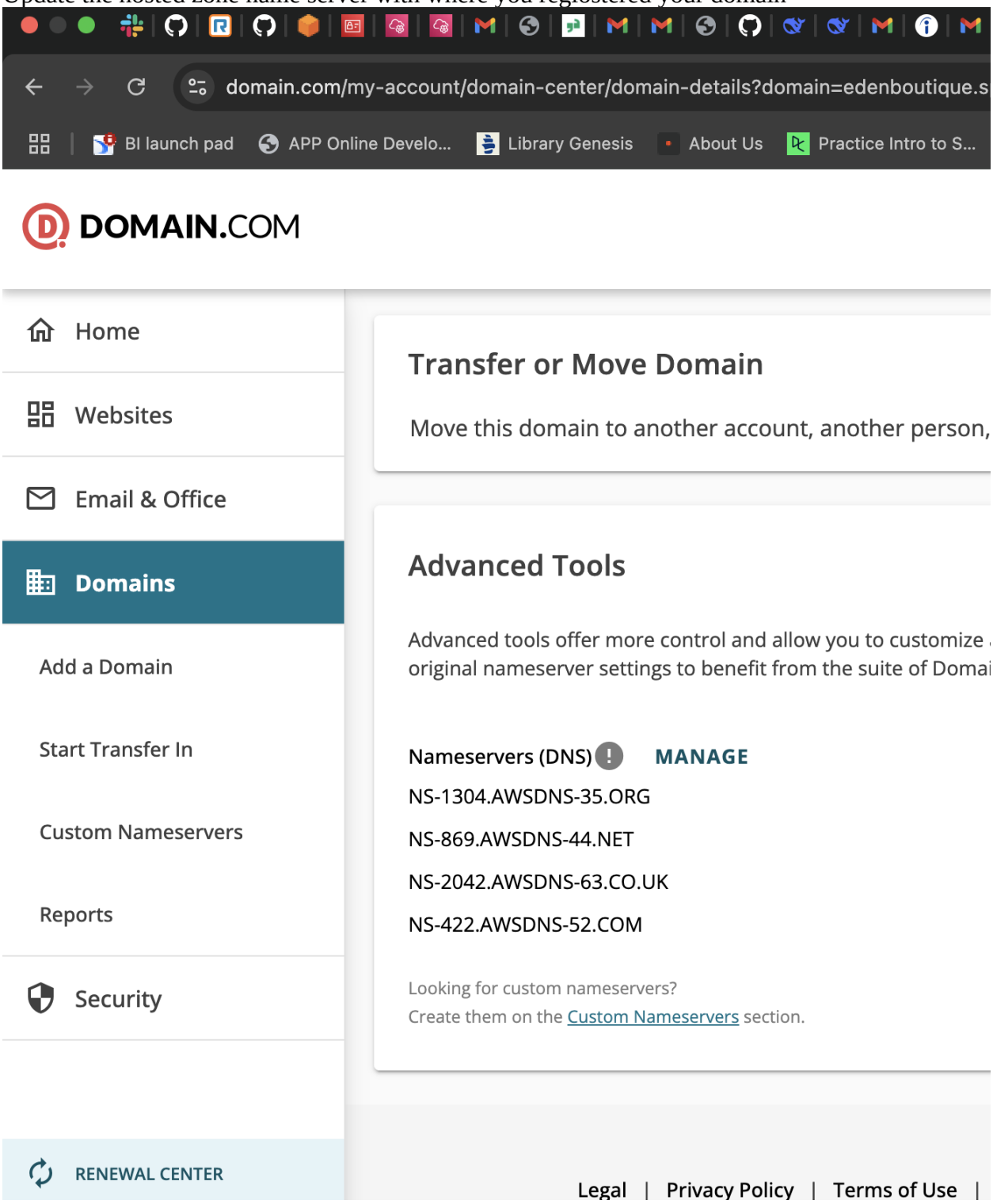
After writing the code for infrastructure deployment, terraform was used to provision the infrastructure.before this provisioning, we set up our domain to be up and running.
First we purchase and registered a domain (edenboutique.space) at www.domain.com



Create a route 53 hosted zone


us-east-1.console.aws.amazon.com/route53/v2/hostedzones?region=us-east-1#ListRecordSets/Z0092163ETAORMR2P7EQ
BI launch pad
APP Online Develo...
Library Genesis
About Us
Practice Intro to S...
general-python | D...
Functions | Python
Kaggle Kernels
Search
[Option+S]
Route 53 > Hosted zones > edenboutique.space
Route 53
Dashboard
Hosted zones
Health checks
Profiles New
IP-based routing
CIDR collections
Traffic flow
Traffic policies
Policy records
Domains
Registered domains
Requests
Resolver
VPCs
Inbound endpoints
Outbound endpoints
Rules
Query logging
Public edenboutique.space Info
Delete zone
Test record
Configure query logging
Hosted zone details
Edit hosted zone
Records (2)
DNSSEC signing
Hosted zone tags (0)
Records (2) Info
Delete record
Import zone file
Create record
Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.
Filter records by property or value
Type
Routing p...
Alias
1
Record name
Type
Routin...
Differ...
Alias
Value/Route traffic to
TTL (s...
Health ...
edenboutique.space
NS
Simple
-
No
ns-1304.awsdns-35.org.
ns-869.awsdns-44.net.
ns-2042.awsdns-63.co.uk.
ns-422.awsdns-52.com.
172800
-
edenboutique.space
SOA
Simple
-
No
ns-1304.awsdns-35.org. aws...
900
-
0 records sele
Select a record to

Update the hosted zone name server with where you regiostered your domain



Next, we provision the s3 bucket as well as the jenkins and vault server. the command to

provision this are all embeded in the s3 create-remote-state script.
On your terminal, run the “create-remote-state.sh” to provision athe vault and Jenkins server

```
26   # # Enable versioning
27   echo "🔄 Enabling versioning for S3 bucket..."
28   if ! aws s3api put-bucket-versioning --bucket "$BUCKET_NAME" --versioning-configuration Status=Enabled --re
29       handle_error "Failed to enable versioning for S3 bucket '$BUCKET_NAME'."
30   fi
31   echo "✅ Versioning enabled successfully."
32
33   echo "🎉 S3 Remote State Management Setup Complete!"
34   echo "🌍 S3 Bucket: $BUCKET_NAME"
35
36   # provision the vault and jenkins server
37   echo "🚀 Provisioning Vault and Jenkins server..."
38   cd ./vault-jenkins
39   terraform init
40   terraform fmt --recursive
41   terraform validate
42   terraform apply -auto-approve
43   terraform output
44   echo "✅ Vault and Jenkins server provisioned successfully."
45
```

1

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

 1487  cd autodiscovery-proj
 1488  cd vault-jenkins
 1489  ssh -i auto-discov-key.pem ec2-user@18.170.218.148\n
 1490  ssh -i auto-discov-key.pem ec2-user@18.170.218.148\n
 1491  brew tap hashicorp/tap\nbrew install hashicorp/tap/terraform
 1492  terraform --version
franklinonyia@Franklins-MacBook-Pro auto-discovery-mayist %
* History restored

The operation couldn't be completed. Unable to locate a Java Runtime.
Please visit http://www.java.com for information on installing Java.

franklinonyia@Franklins-MacBook-Pro auto-discovery-mayist % cd autodiscovery-proj
franklinonyia@Franklins-MacBook-Pro autodiscovery-proj % sh create-remote-state.sh
```

2

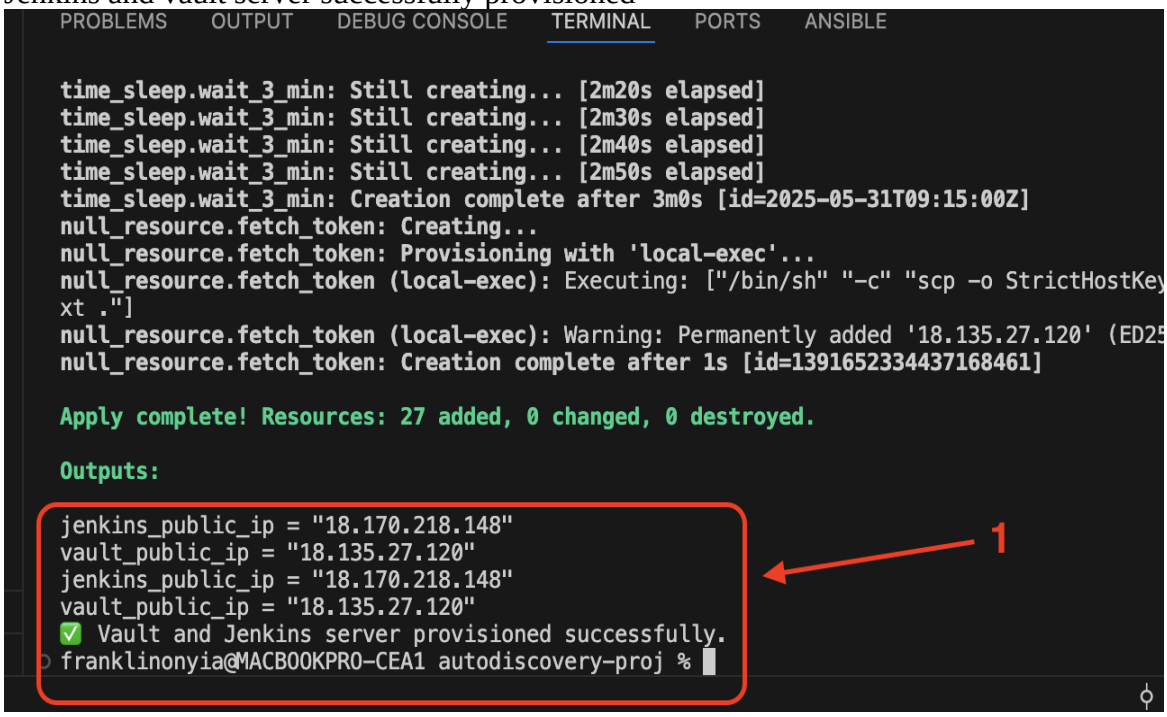
Restart Vis

Bucket, vault and Jenkins provision ongoing

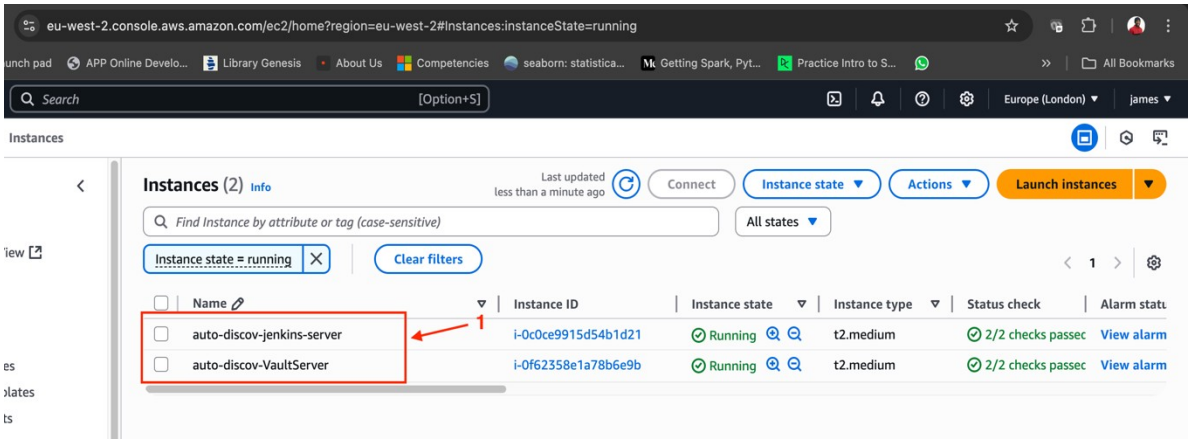
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ANSIBLE

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/onyiafranklin/autodiscovery-proj.git
   2a48643..689f8eb  feature/ebuka -> feature/ebuka
franklinonyia@MACBOOKPRO-CEA1 autodiscovery-proj % sh create-remote-state.sh
🚀 Creating S3 bucket: bucket-pet-adoption...
http://bucket-pet-adoption.s3.amazonaws.com/
✅ S3 bucket 'bucket-pet-adoption' created successfully.
🔄 Enabling versioning for S3 bucket...
✅ Versioning enabled successfully.
🎉 S3 Remote State Management Setup Complete!
🌍 S3 Bucket: bucket-pet-adoption
🚀 Provisioning Vault and Jenkins server...
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/local from the dependency lock file
- Reusing previous version of hashicorp/null from the dependency lock file
- Reusing previous version of hashicorp/tls from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/time from the dependency lock file
- Using previously-installed hashicorp/null v3.2.4
- Using previously-installed hashicorp/tls v4.1.0
- Using previously-installed hashicorp/aws v5.98.0
```

onyiafranklin (1 week

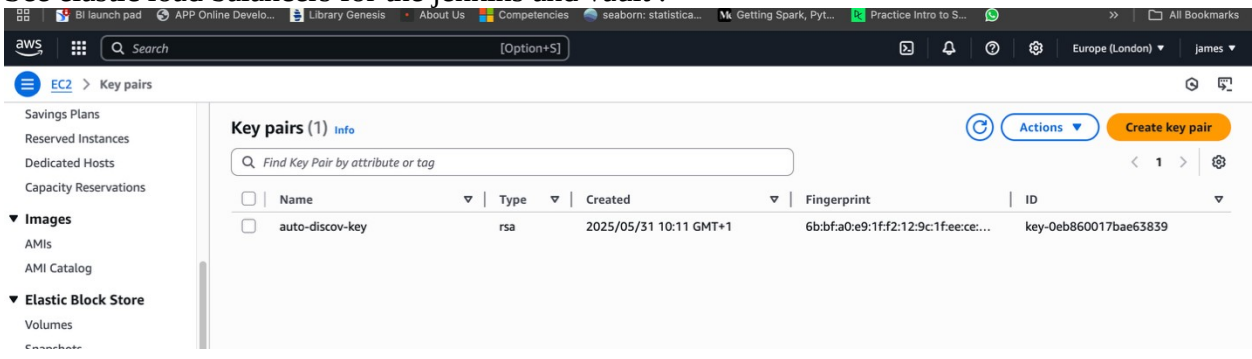
1. Jenkins and vault server successfully provisioned

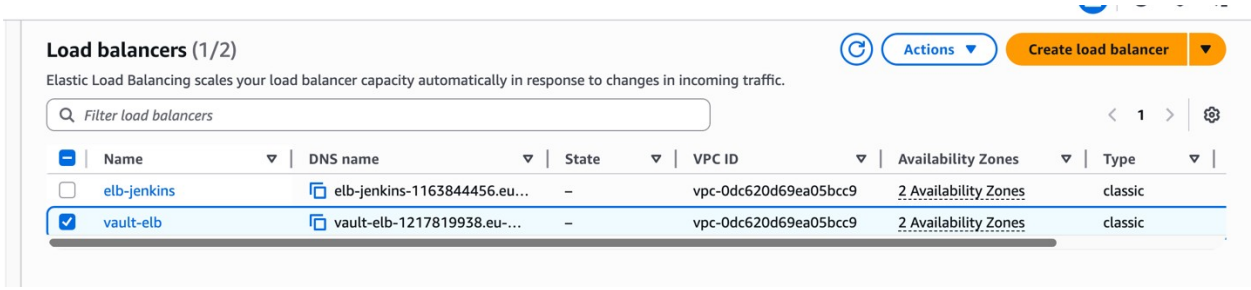


view Jenkins and vault server on the console

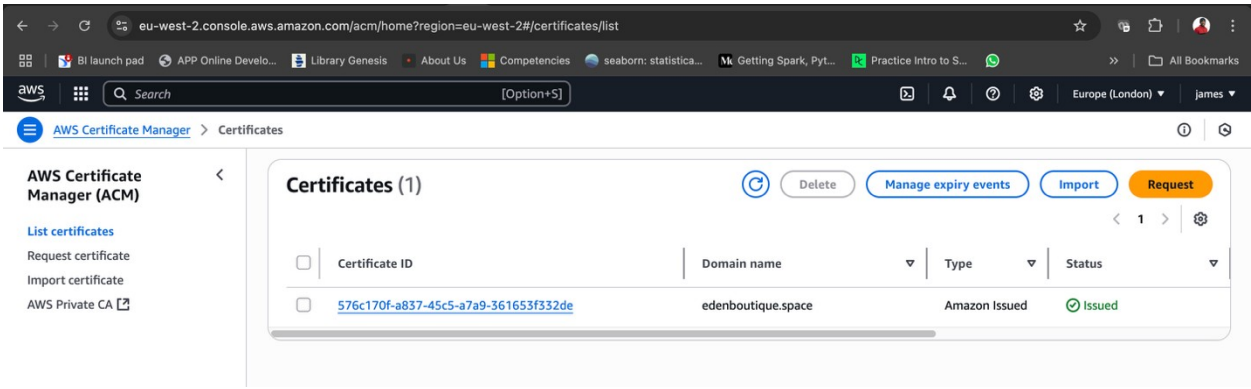


See elastic load balancers for the jenkins and vault .

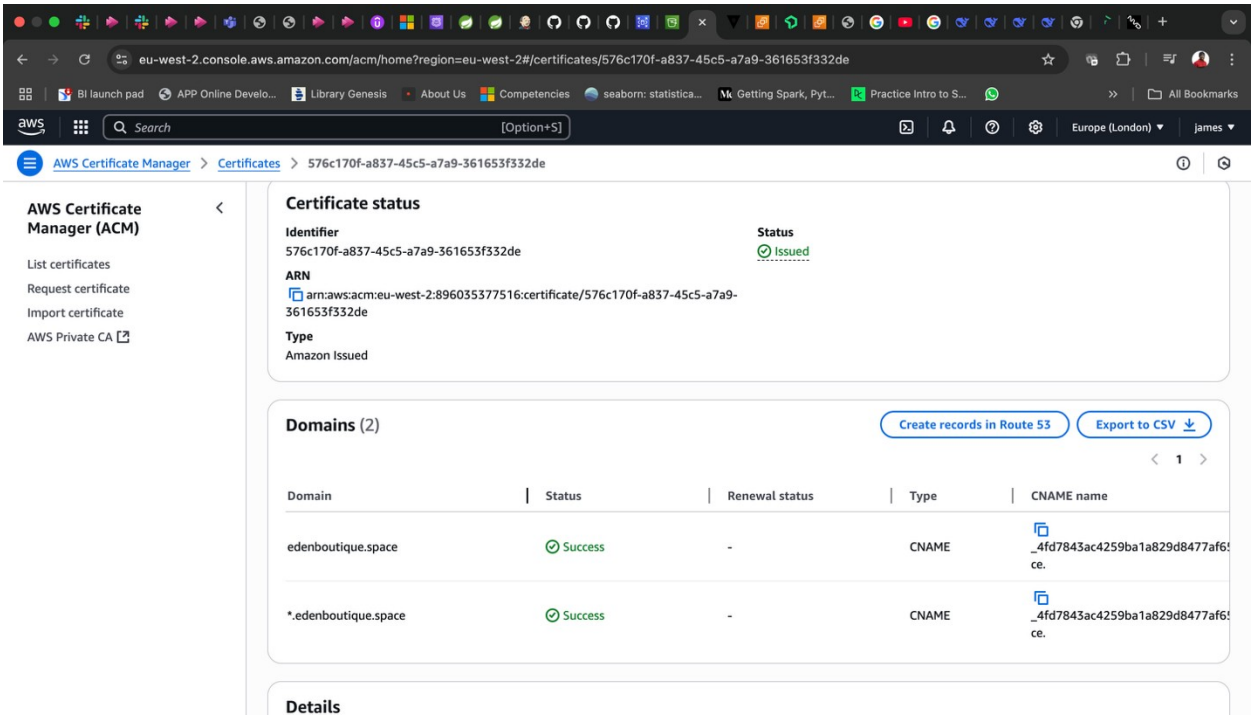




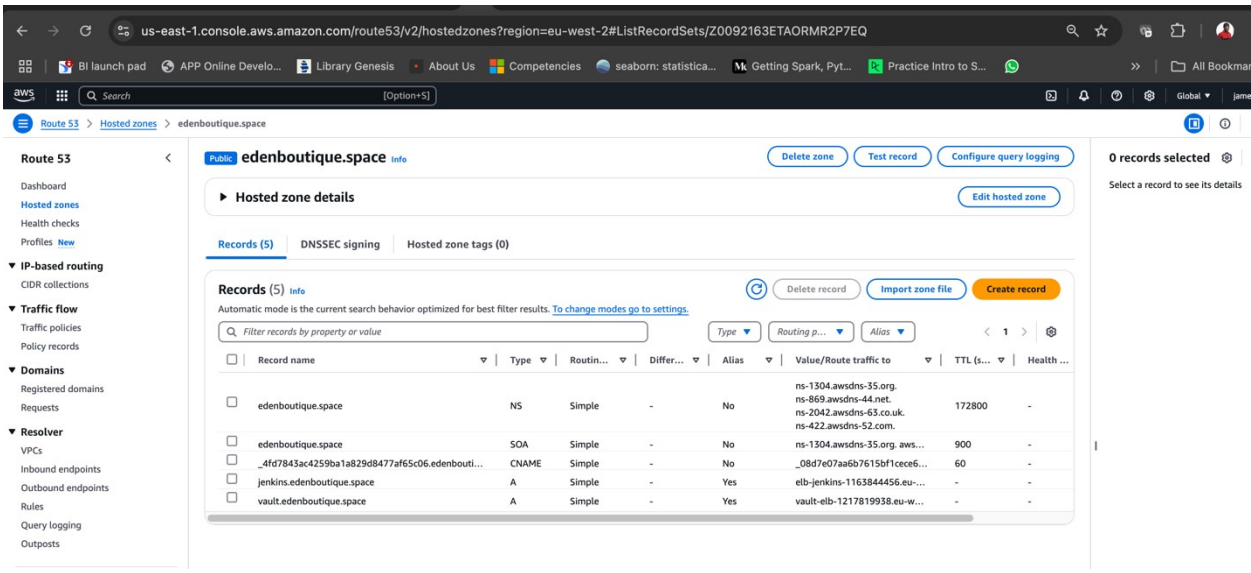
view issued domain certificate



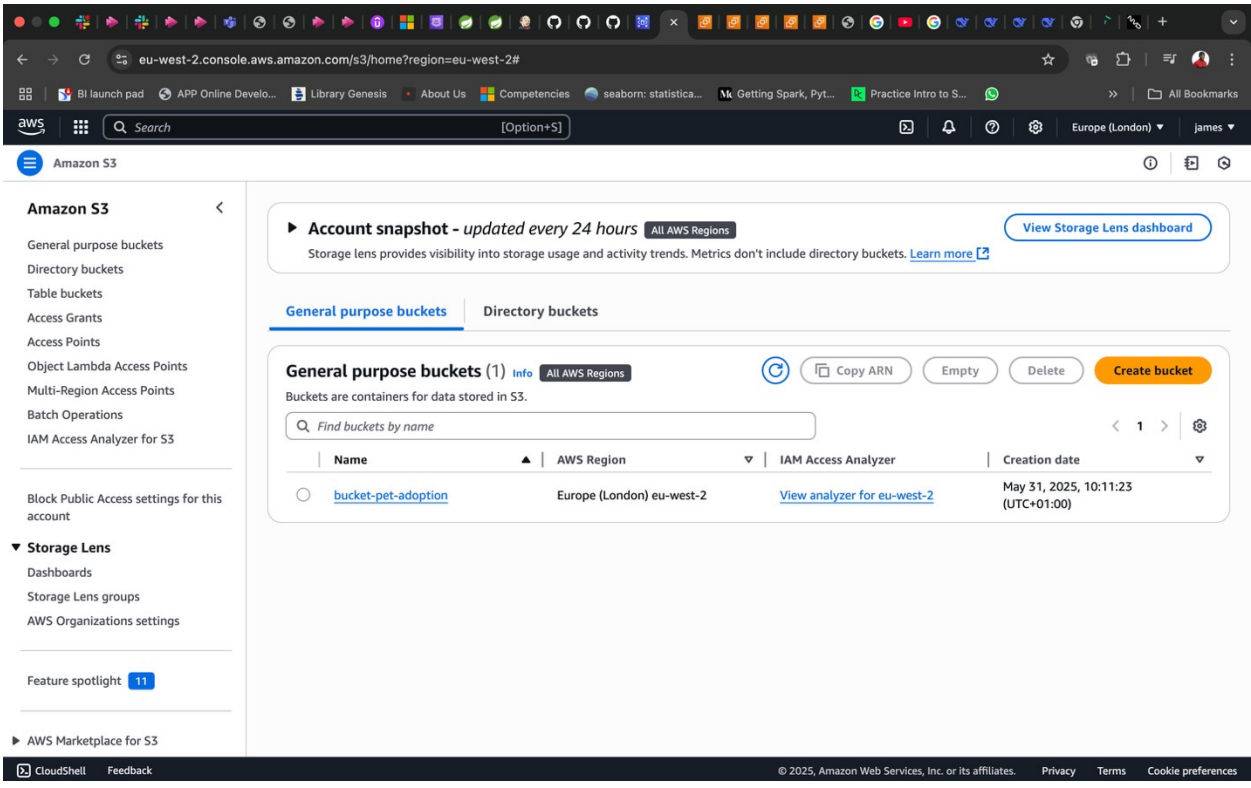
see certificate status for domain and sub domain on console.



view records created for vault and Jenkins



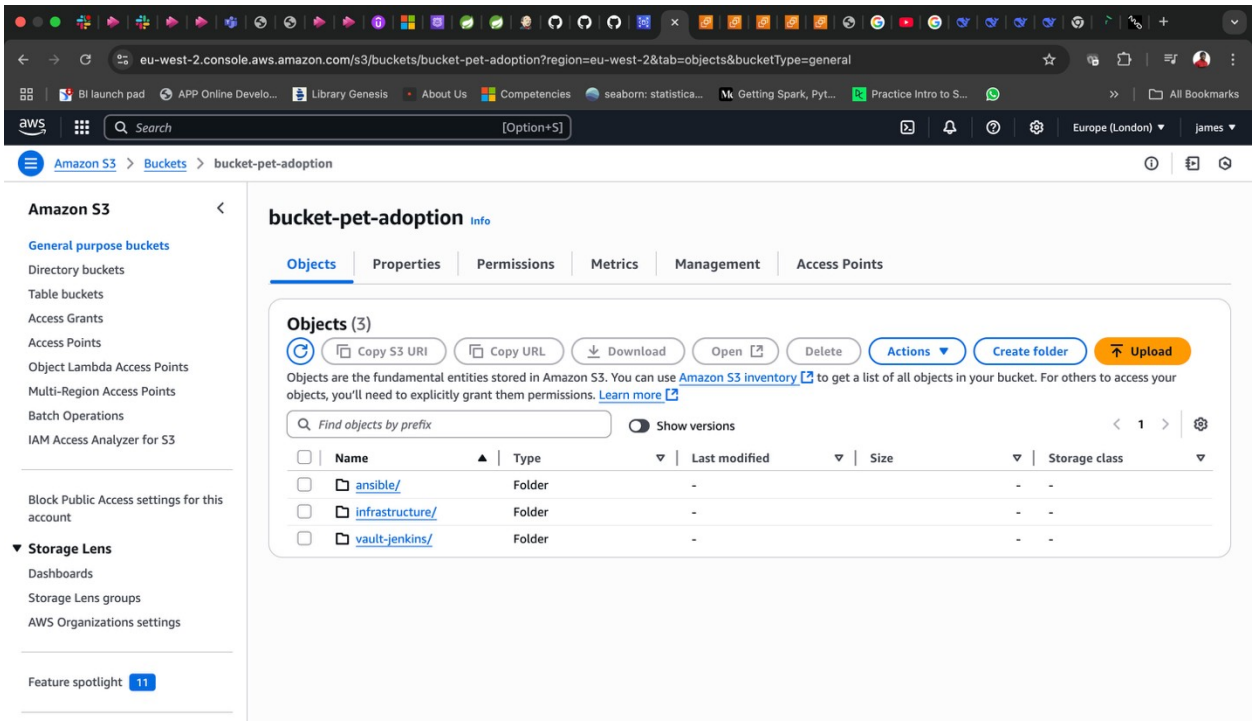
View the created bucket



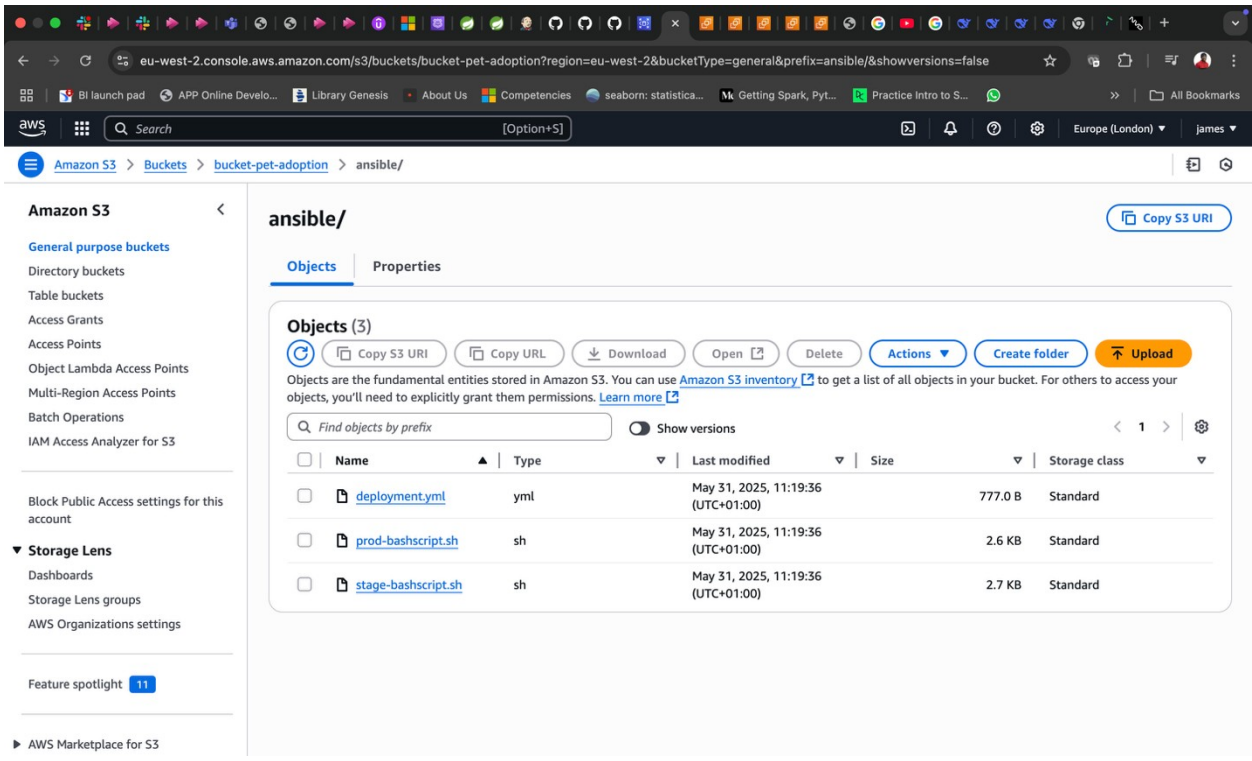
view the directory in the bucket.

Ansible file path :this houses all our ansible deployment file, stage and prod script copied

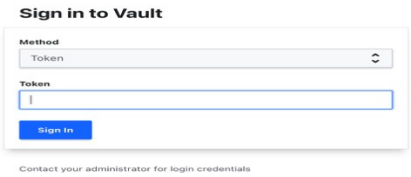
Infrastructure file path:houses the terraform state file



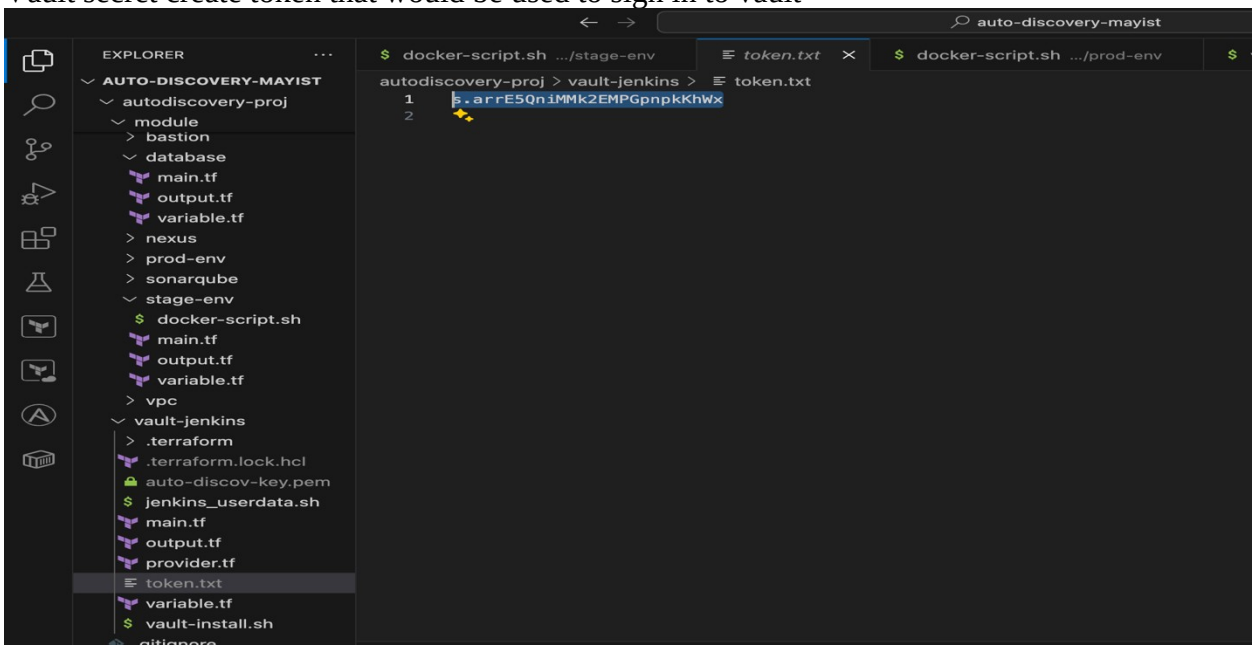
content of the ansible folder



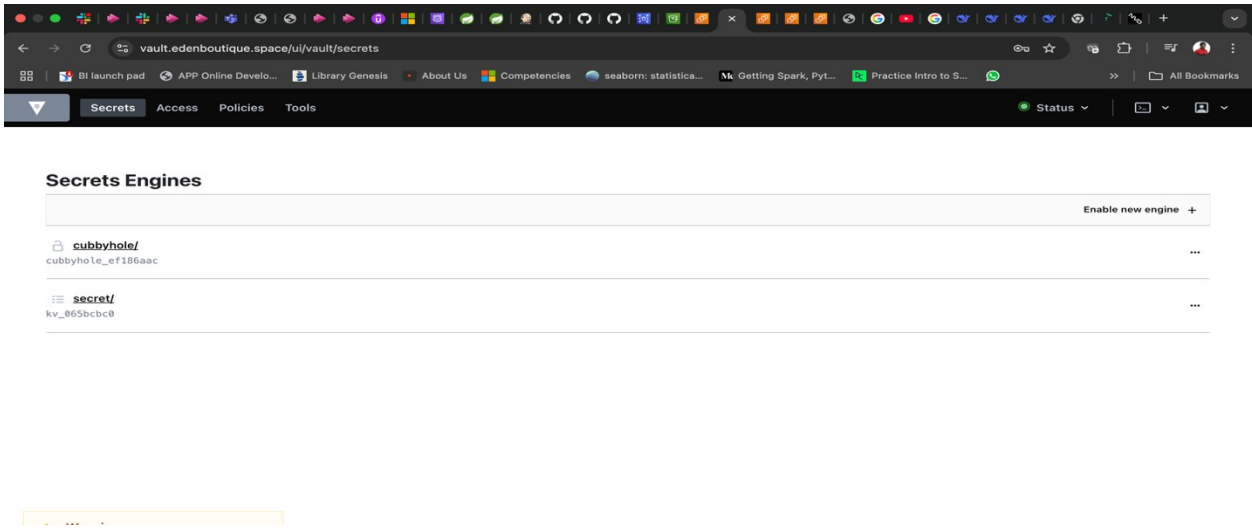
vault sign in

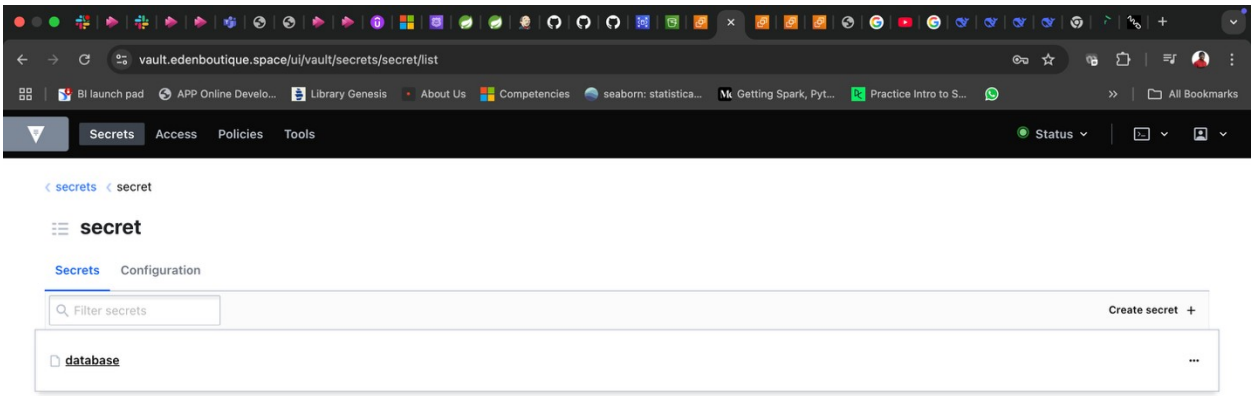


Vault secret create token that would be used to sign in to vault

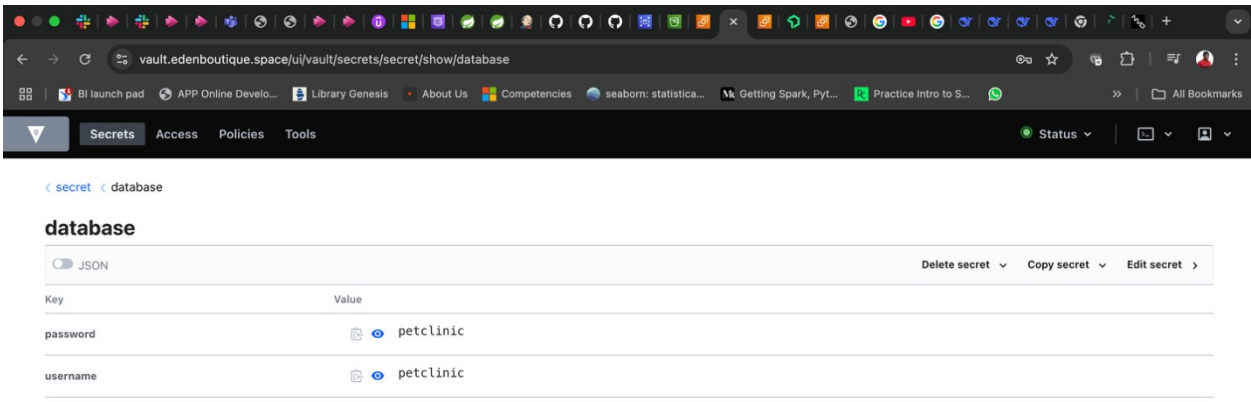


view the secret folder where vault stored our database credentials. Open the secret to view the db credentials

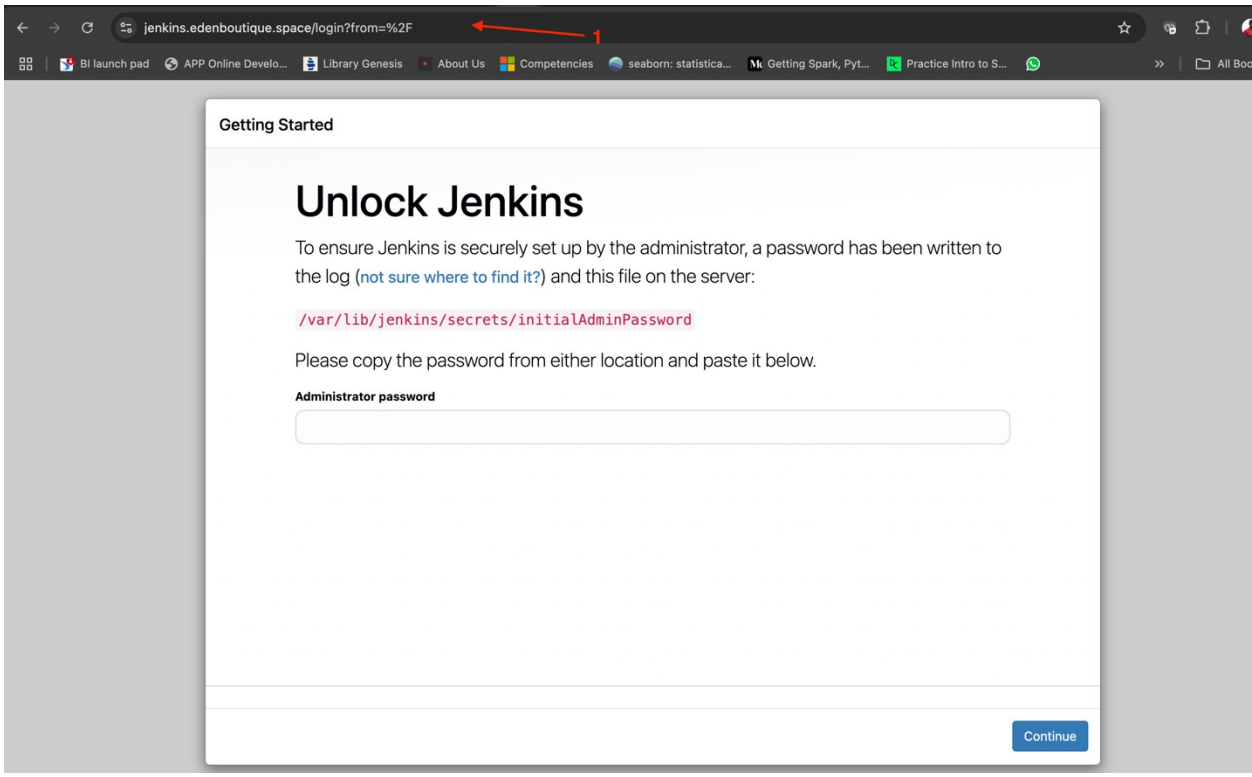




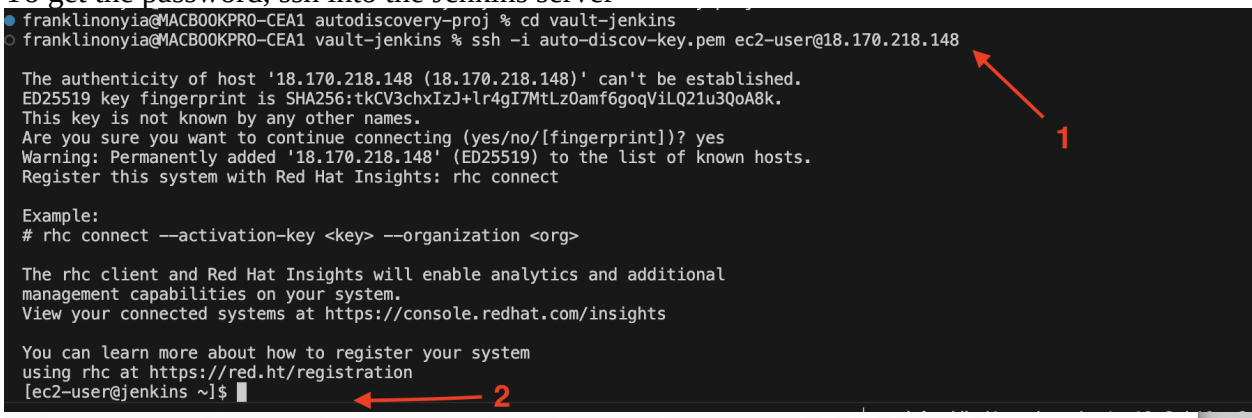
view secrets



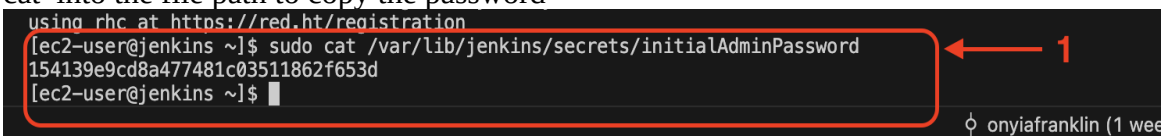
Next is to open the Jenkins using the url “Jenkins.edenboutique.space” the login to Jenkins server to get the admin password on the /var/lib/Jenkins/secret/initialAdminpassword file path



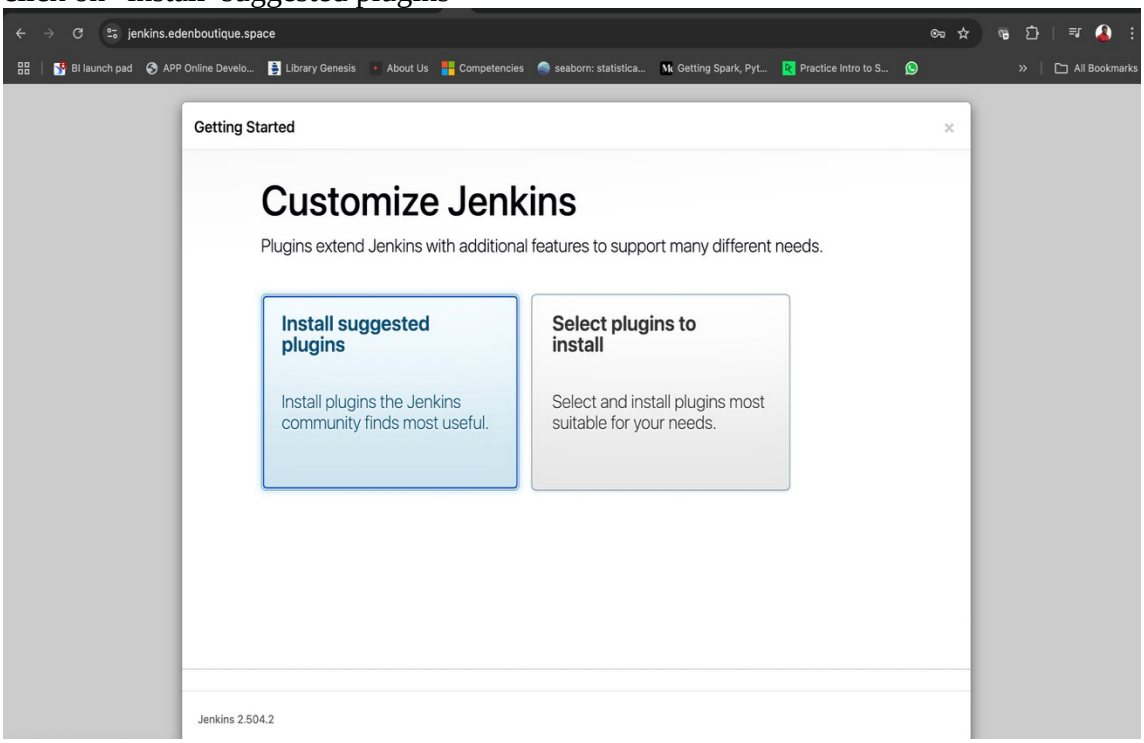
To get the password, ssh into the Jenkins server

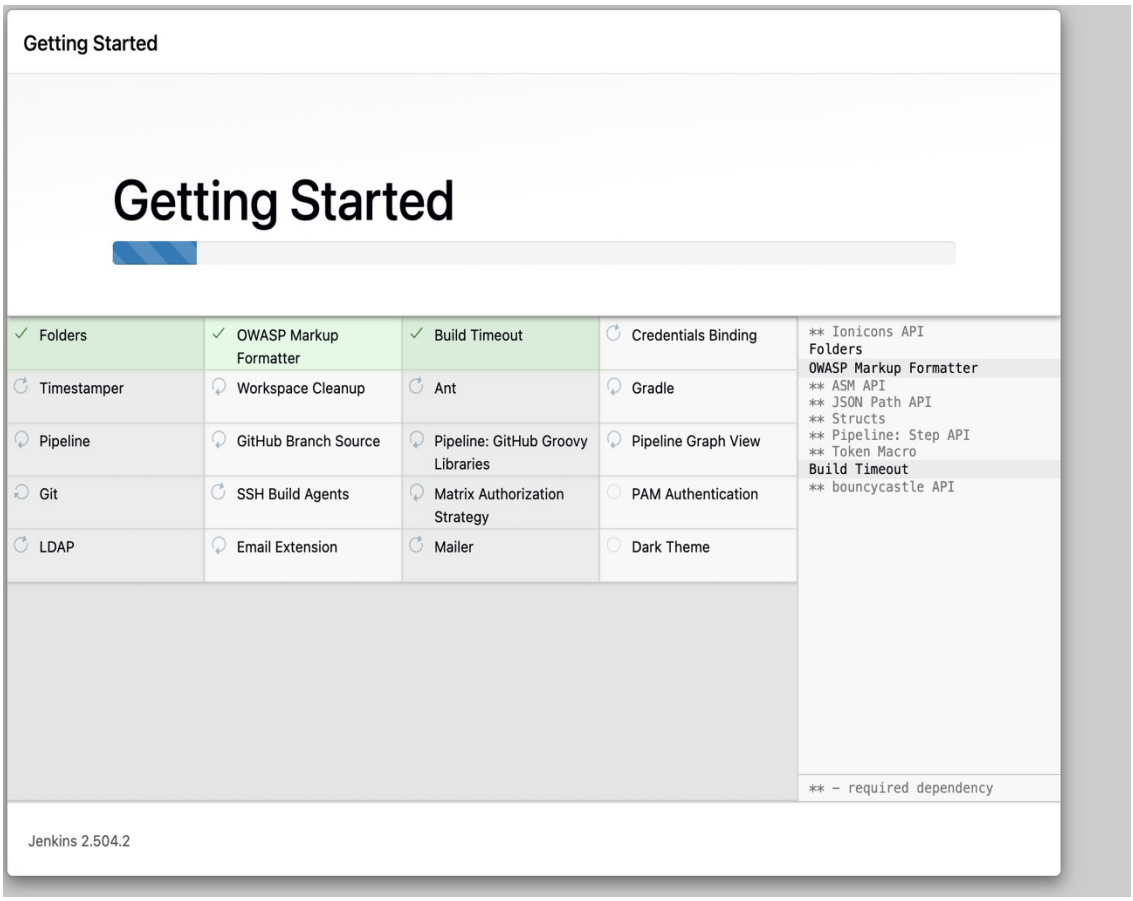


cat into the file path to copy the password

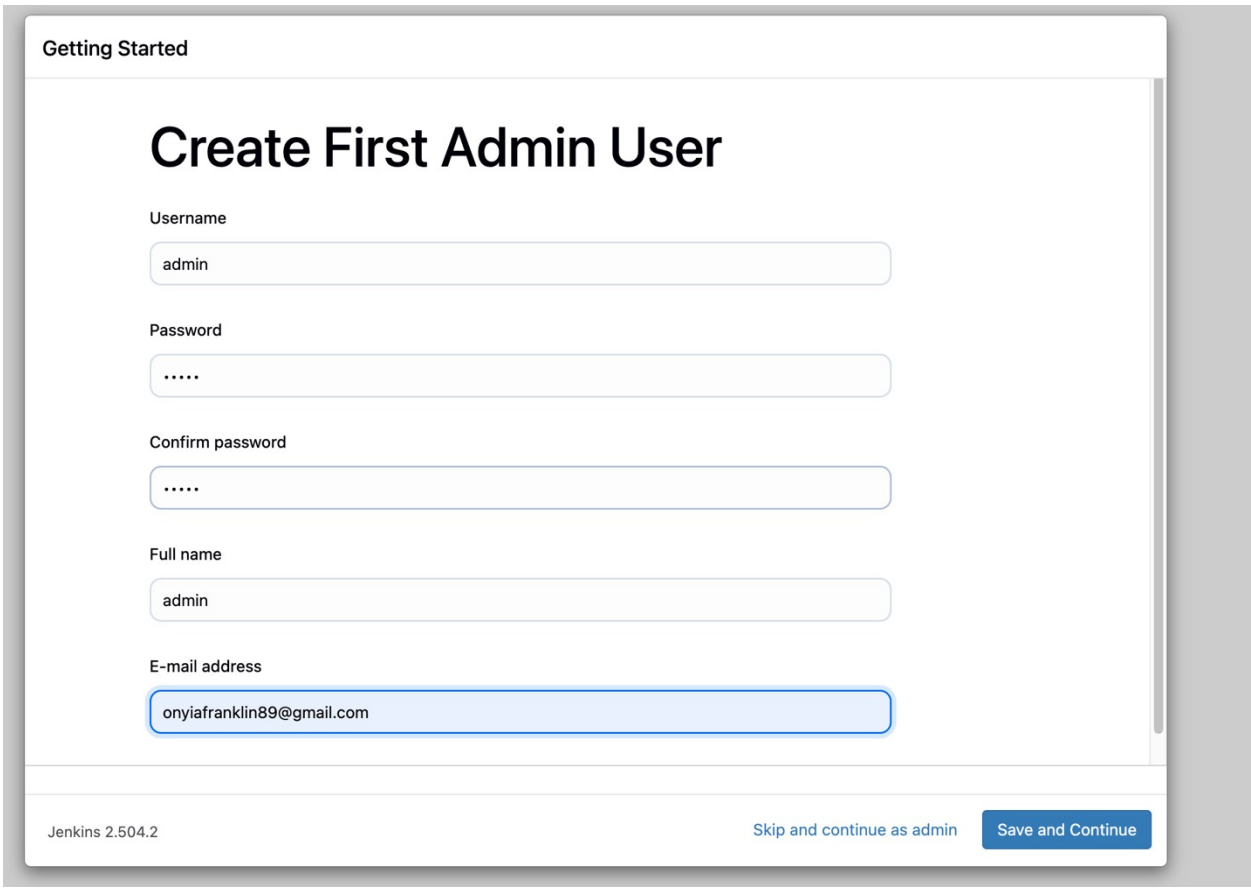


paste then you would be prompted to install suggested plugins
click on “install suggested plugins”

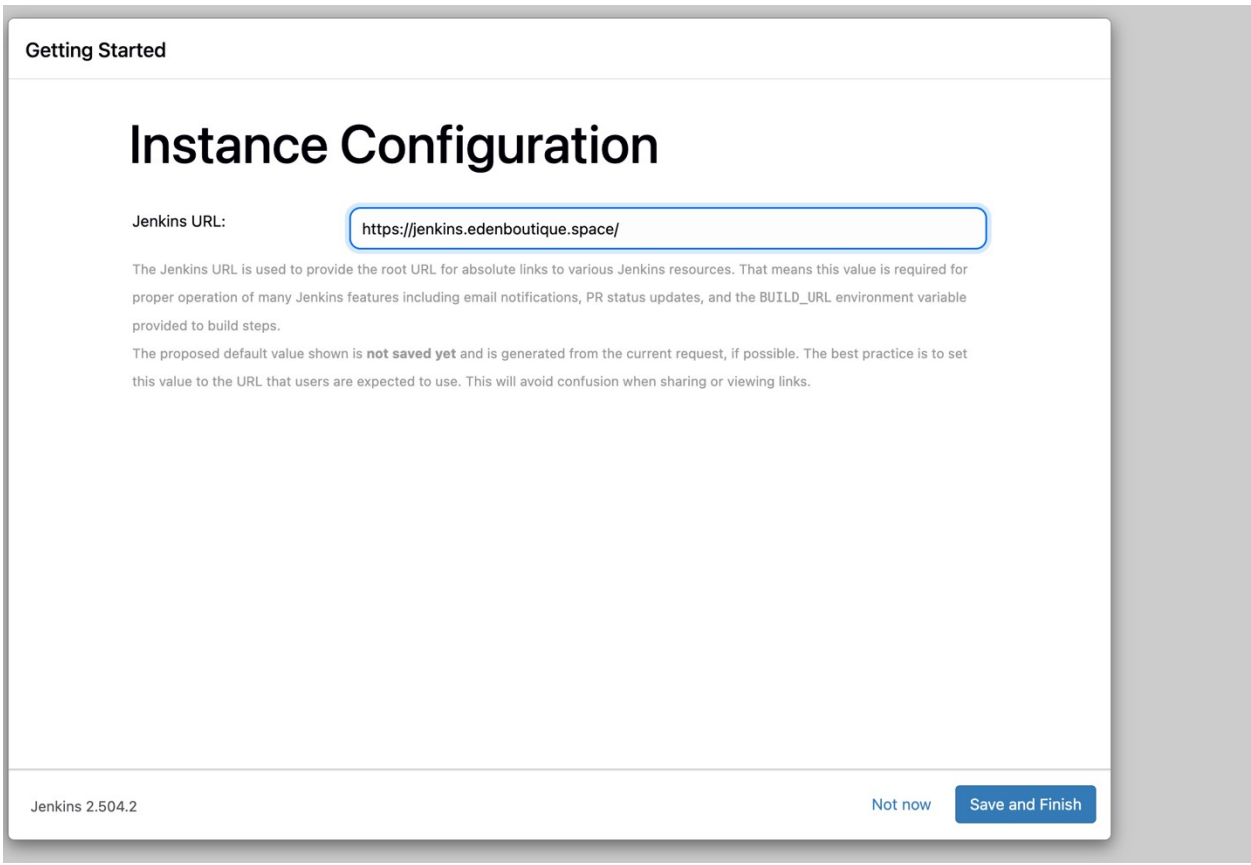




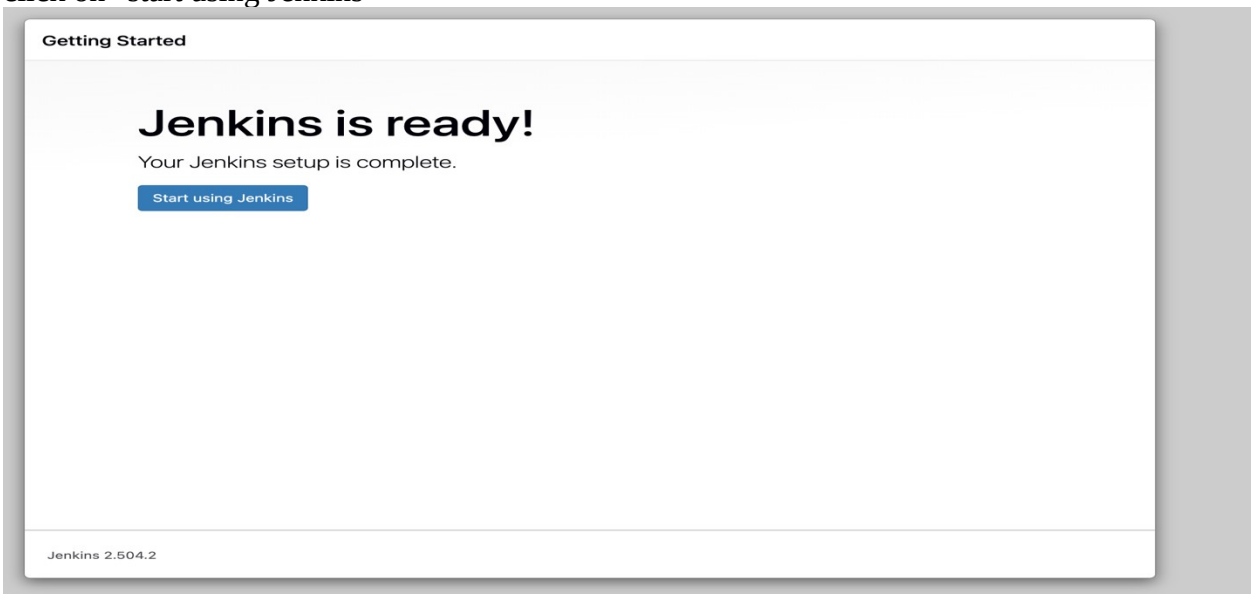
fill in the blank spaces to create first admin user



click on save and finish



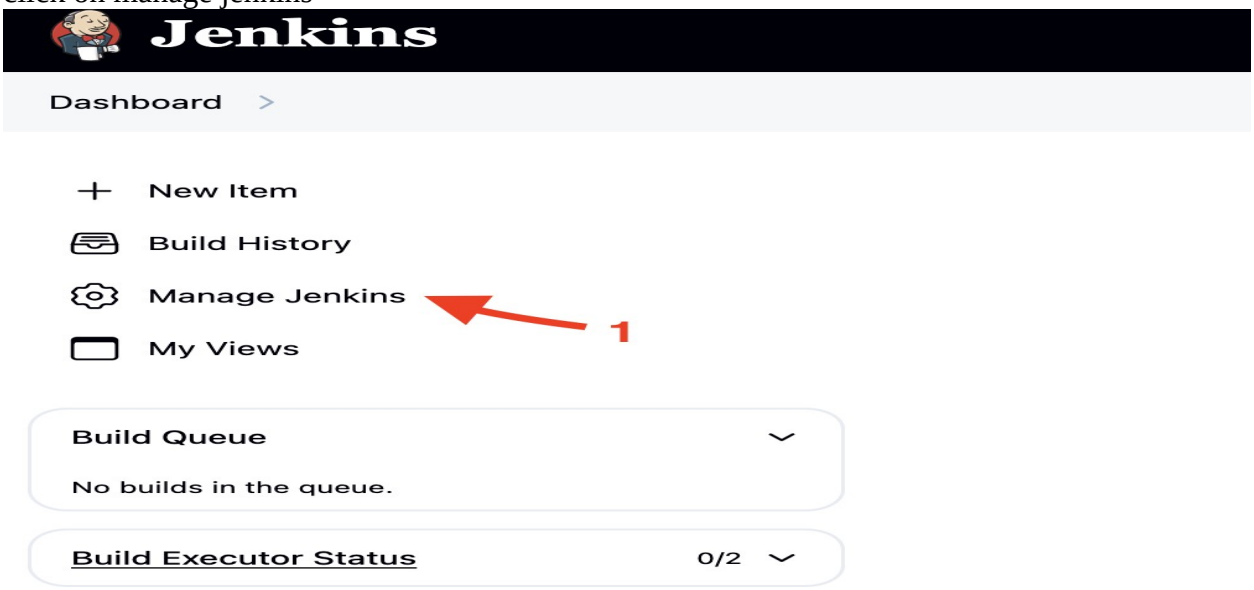
click on “start using Jenkins”



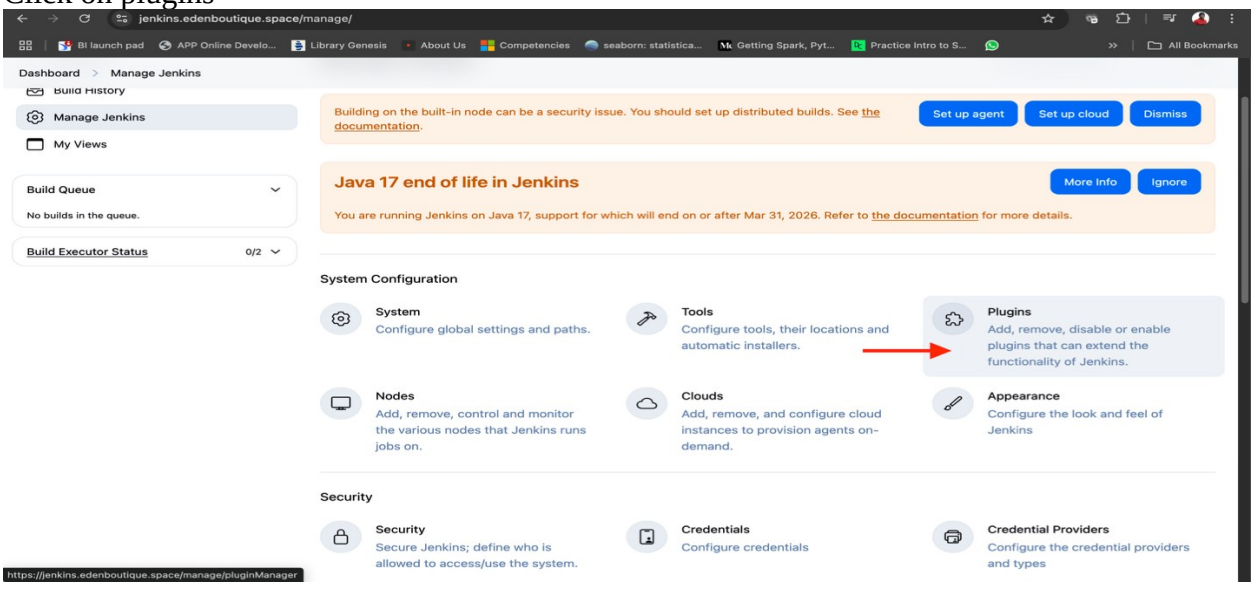
H. INFRASTRUCTURE SETUP AND PROVISIONING USING JENKINS PIPELINE

Here we would setup Jenkins to deploy infrastructure

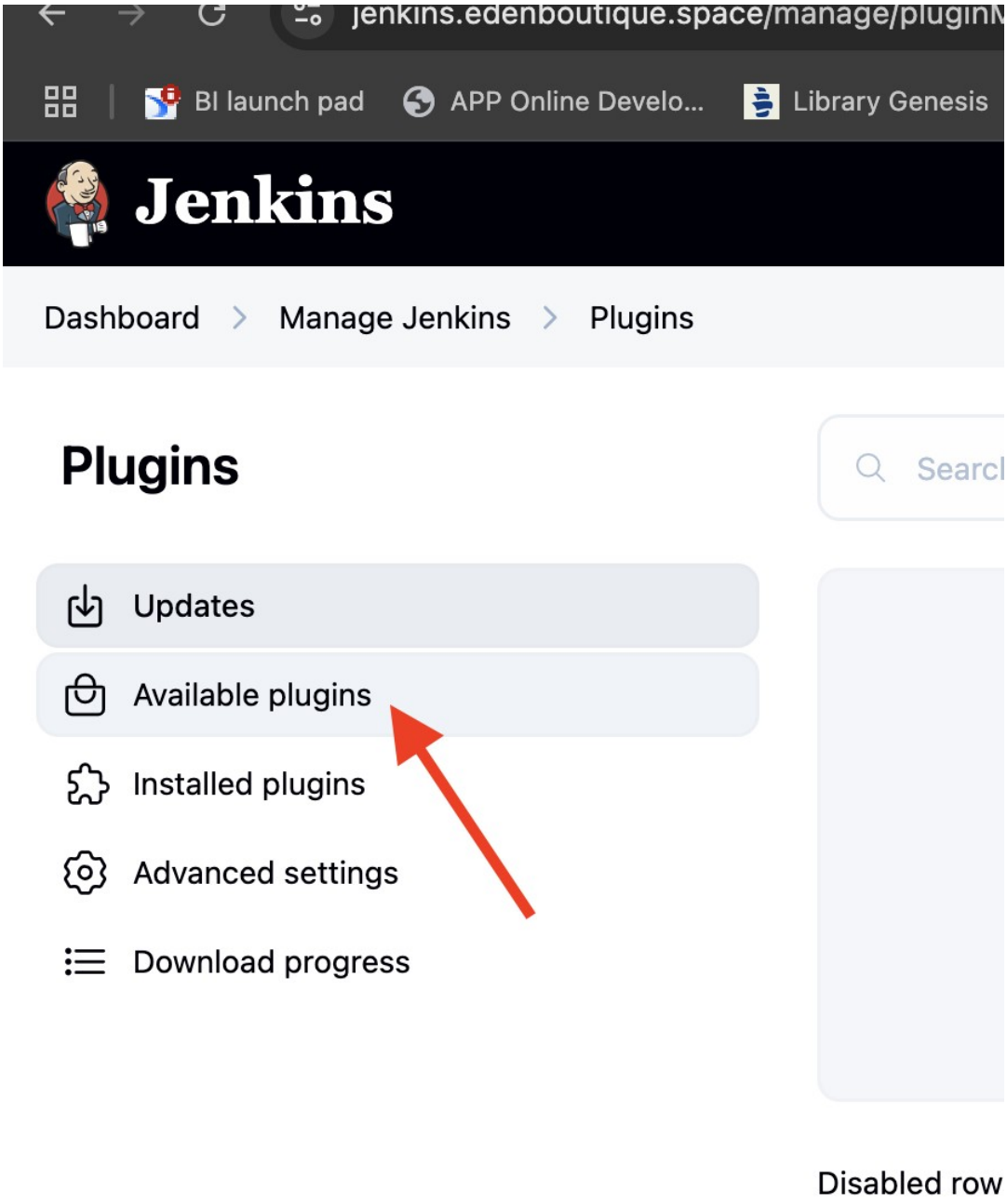
click on manage jenkins



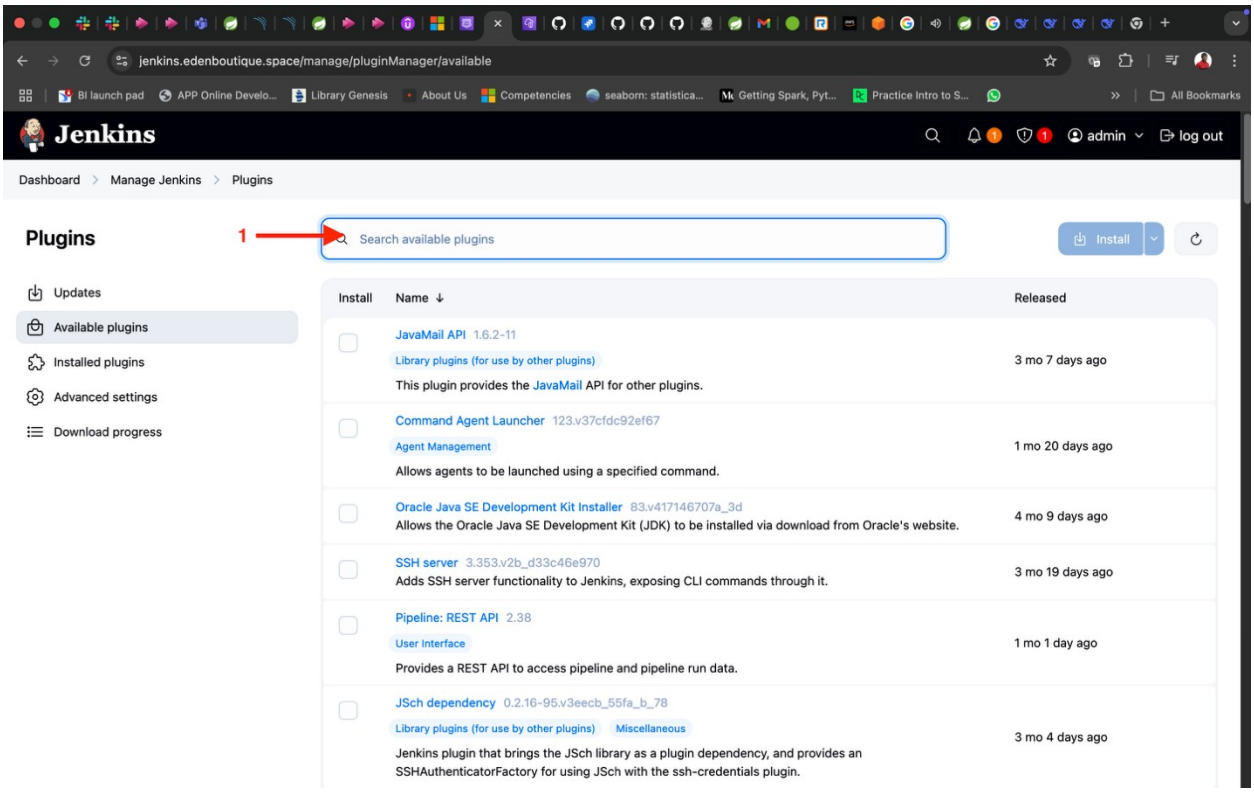
Click on plugins



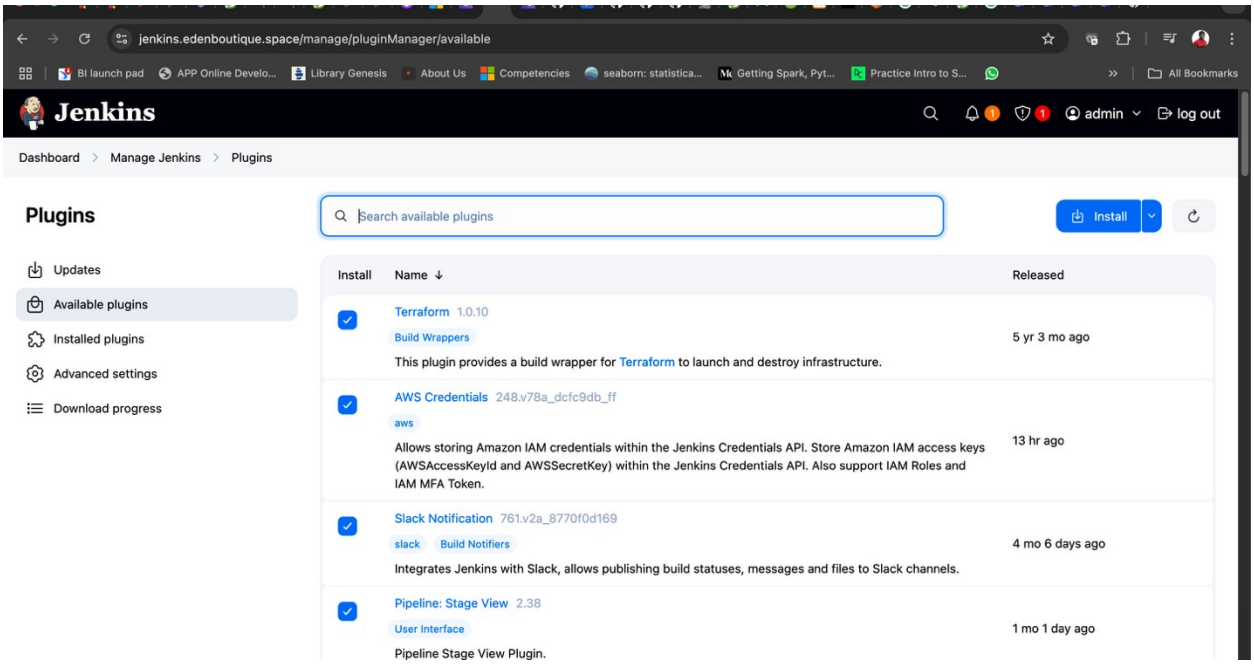
Click on available plugins


jenkins.edenboutique.space/manage/pluginM
BI launch pad
APP Online Develo...
Library Genesis
Jenkins
Dashboard
Manage Jenkins
Plugins
Plugins
Search
Updates
Available plugins
Installed plugins
Advanced settings
Download progress
Disabled row

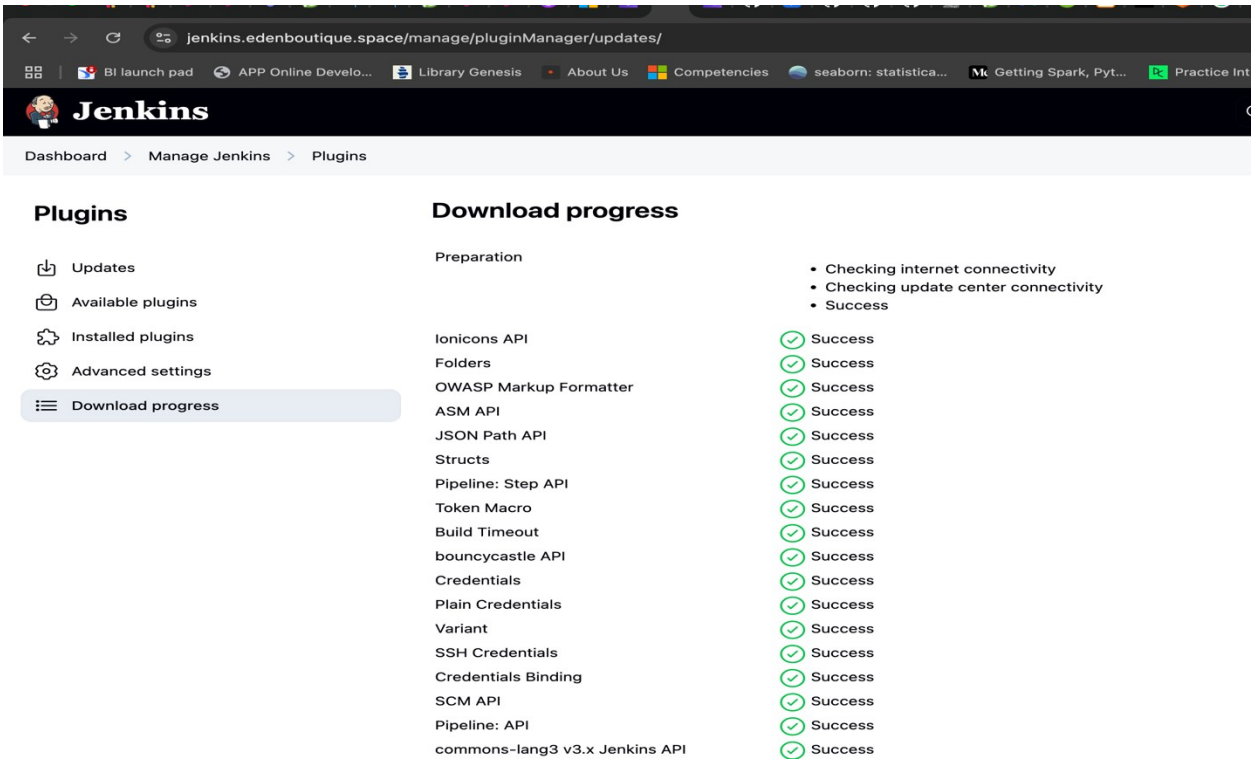
Click on the search button



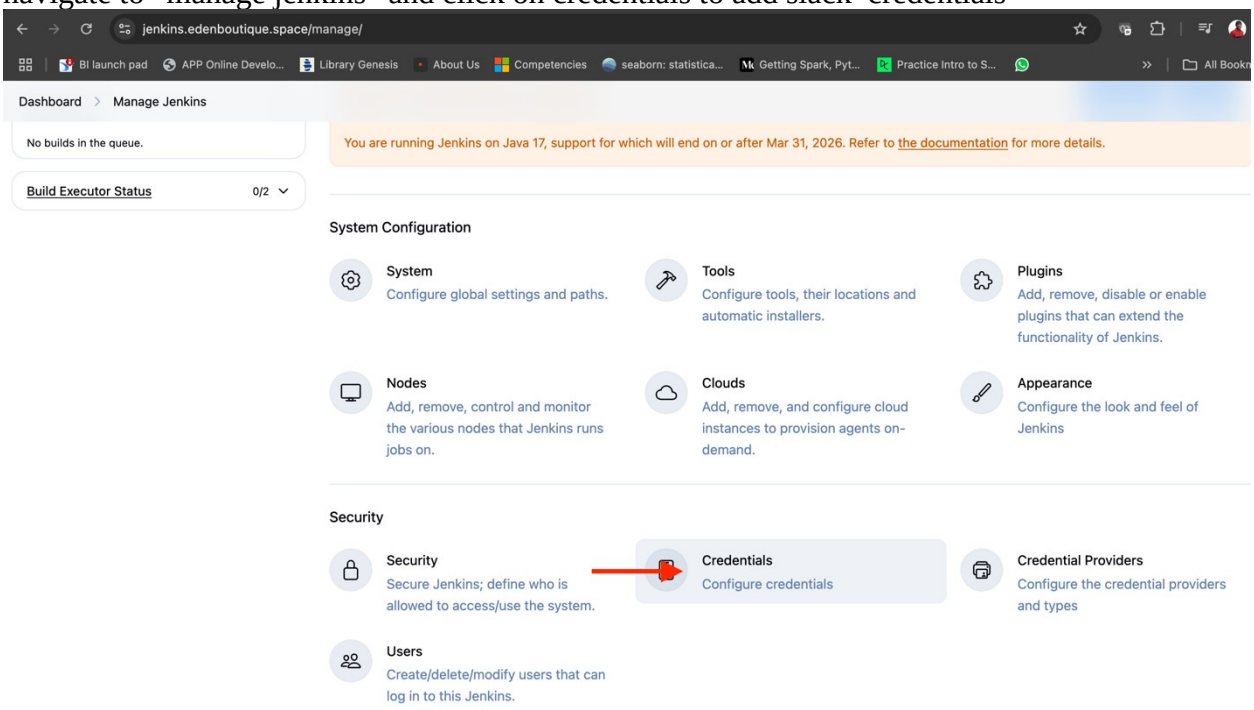
Select terraform, AWS credentials, slack notification and Pipeline Stage View and click on install



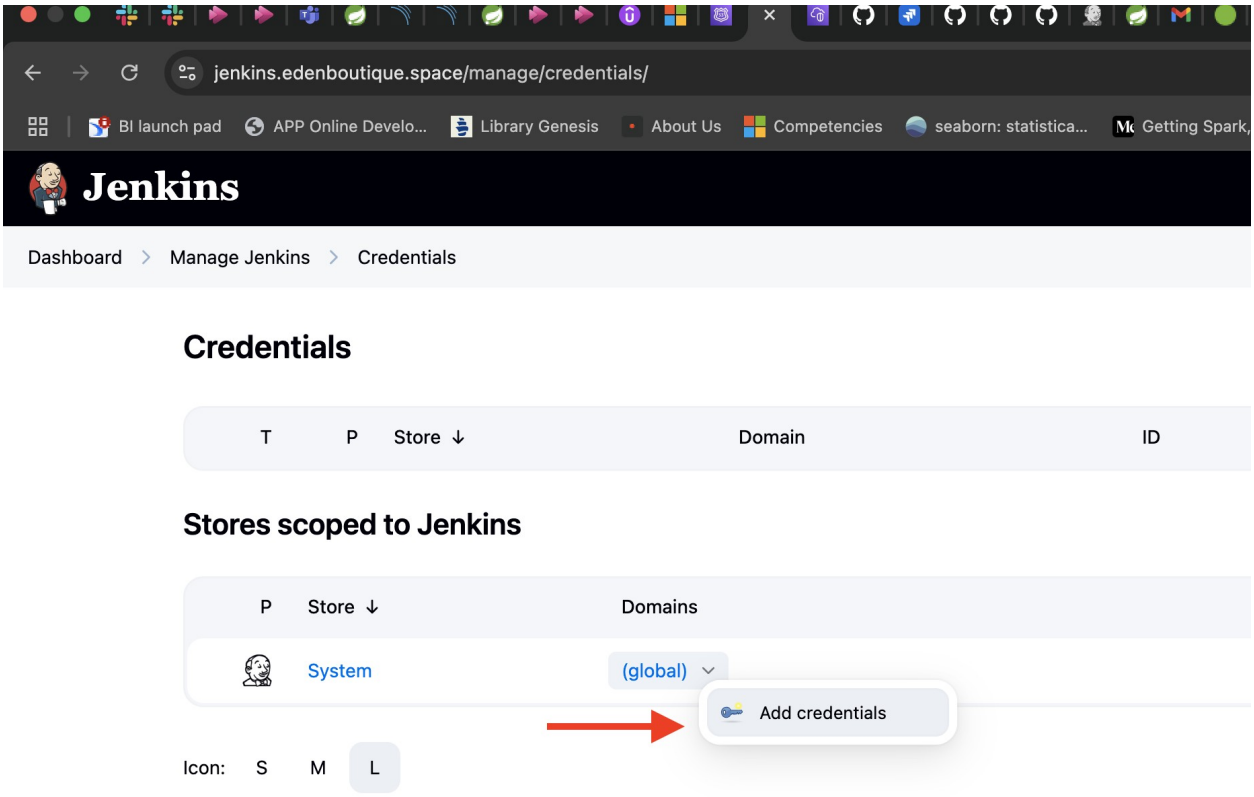
installation ongoing



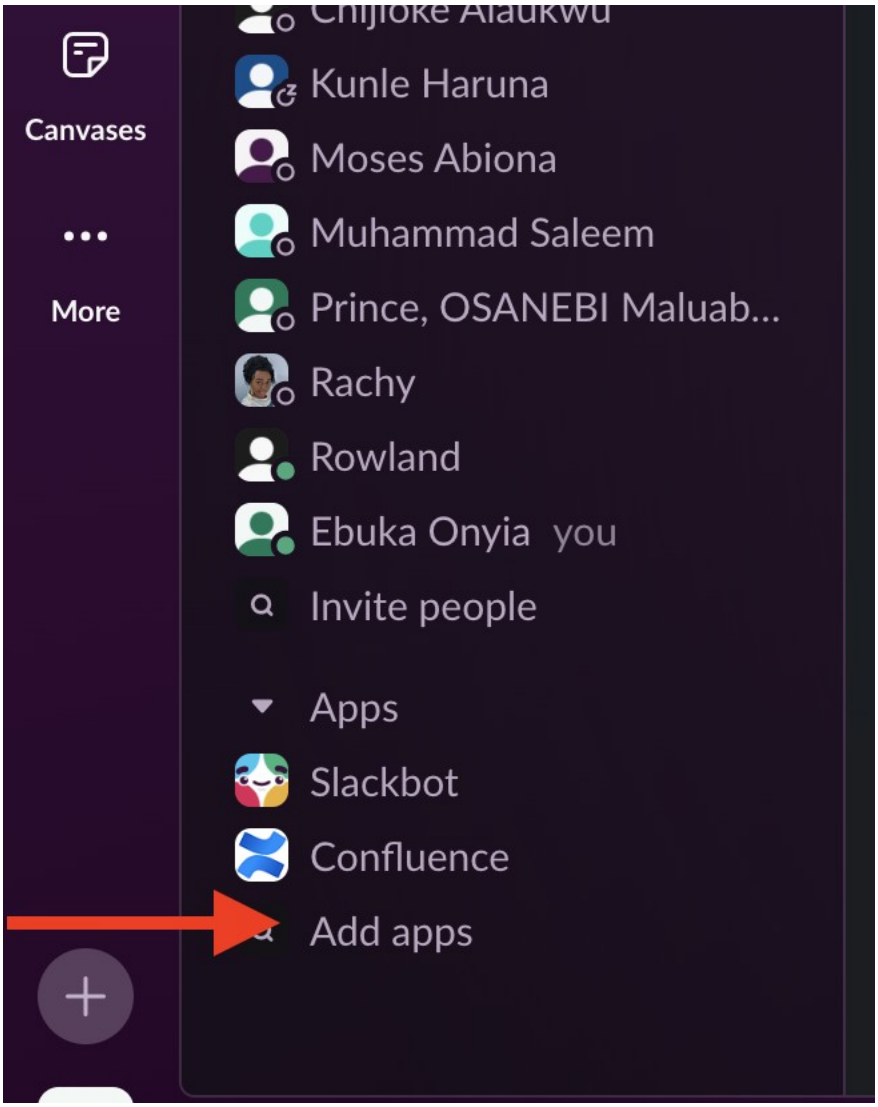
navigate to “manage jenkins” and click on credentials to add slack credentials



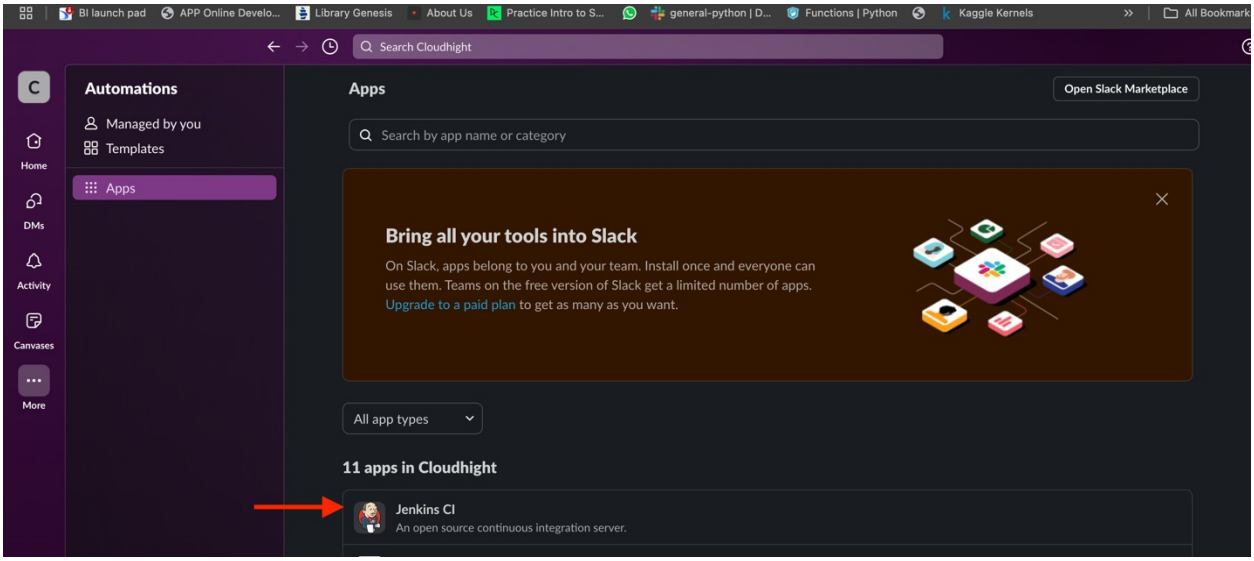
click on global



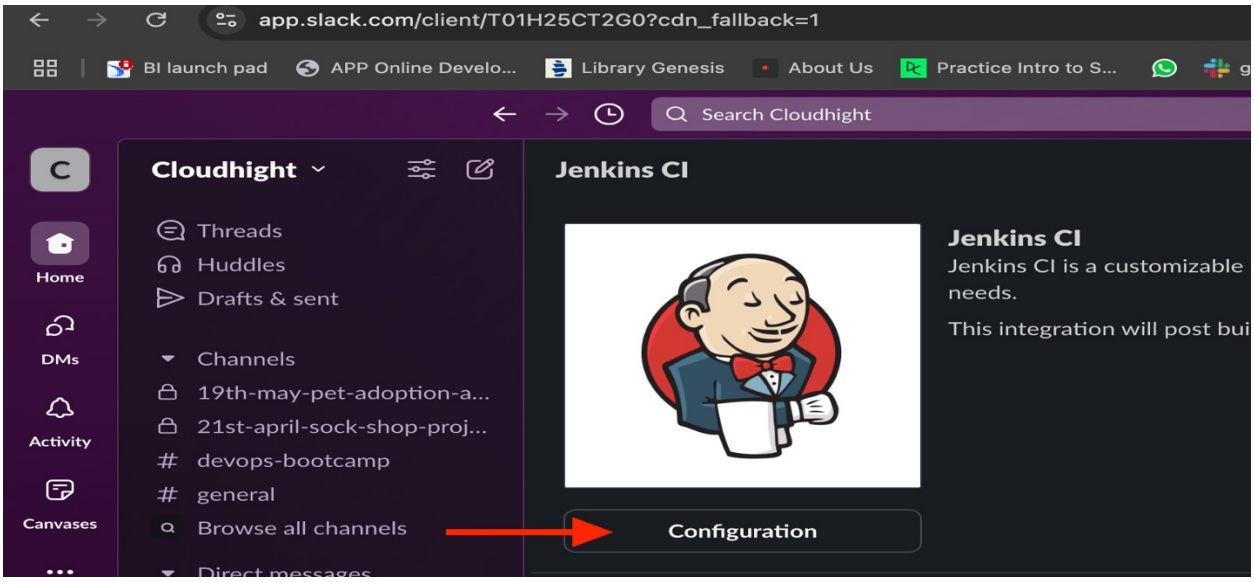
we need to generate a secret for our slack so we can add the credentials to slack. Inother to get thesecret/token, Go to your slack scroll down and click on “Add apps”



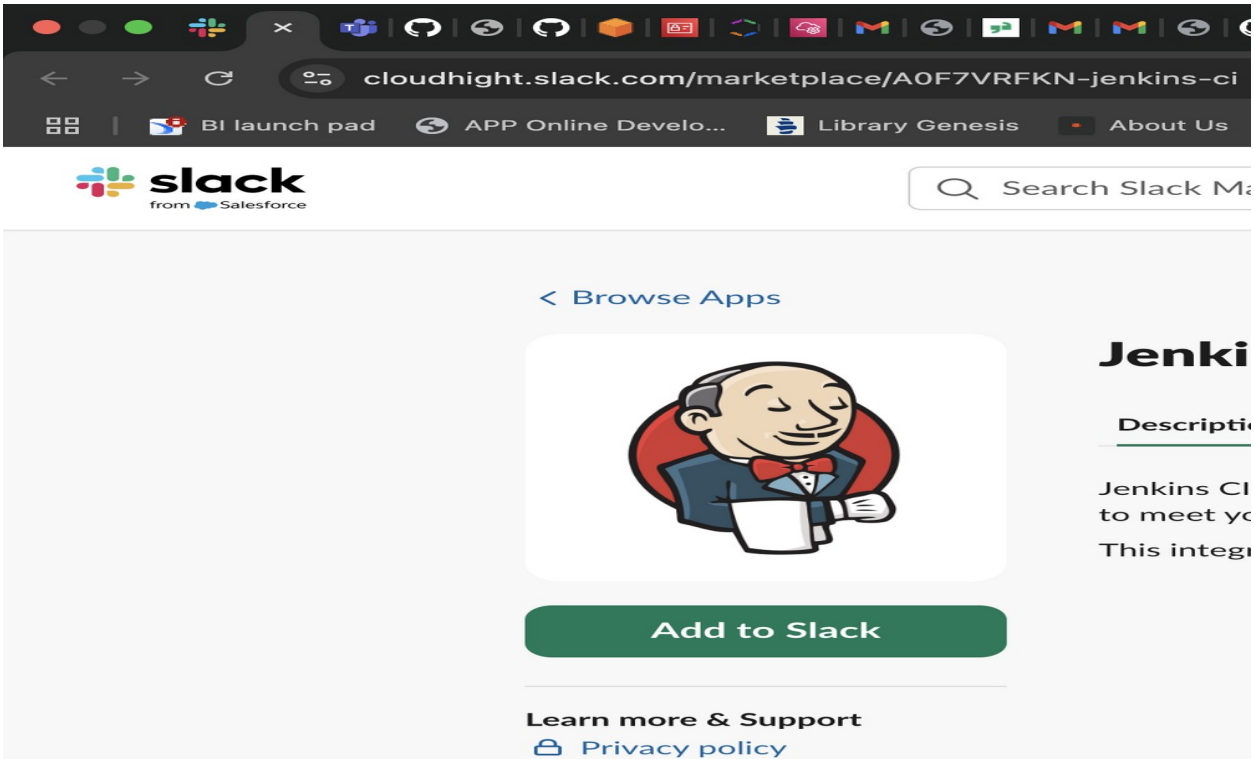
Click on Jenkins CI



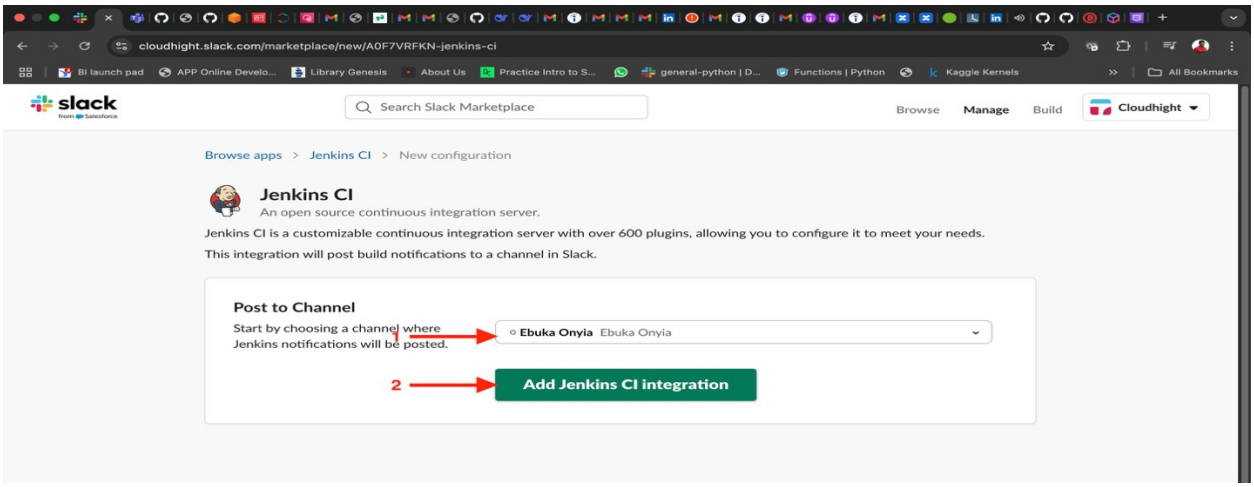
Click on Configuration



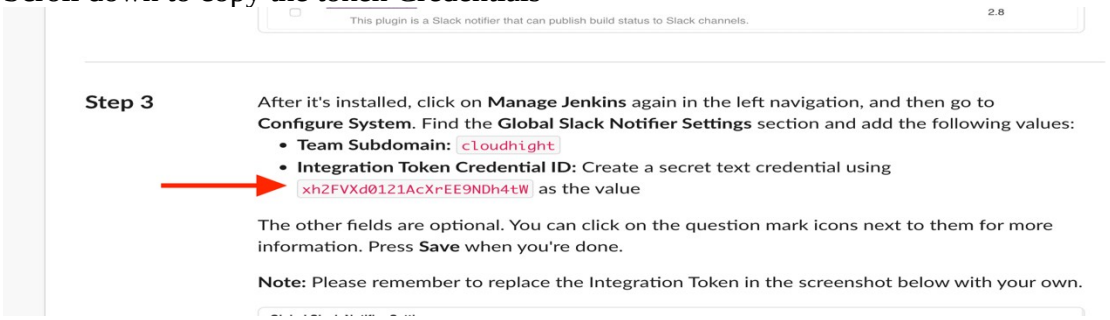
click on add to slack



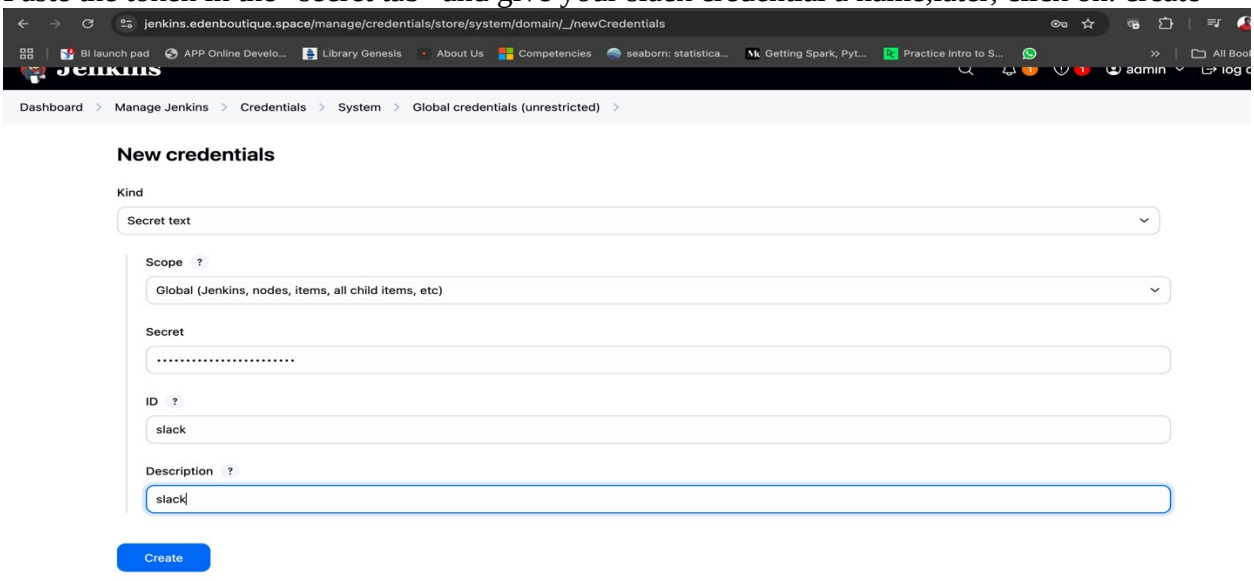
Select the channel to post the notification. Here I selected my name and clicked on “Add Jenkins CI integration”

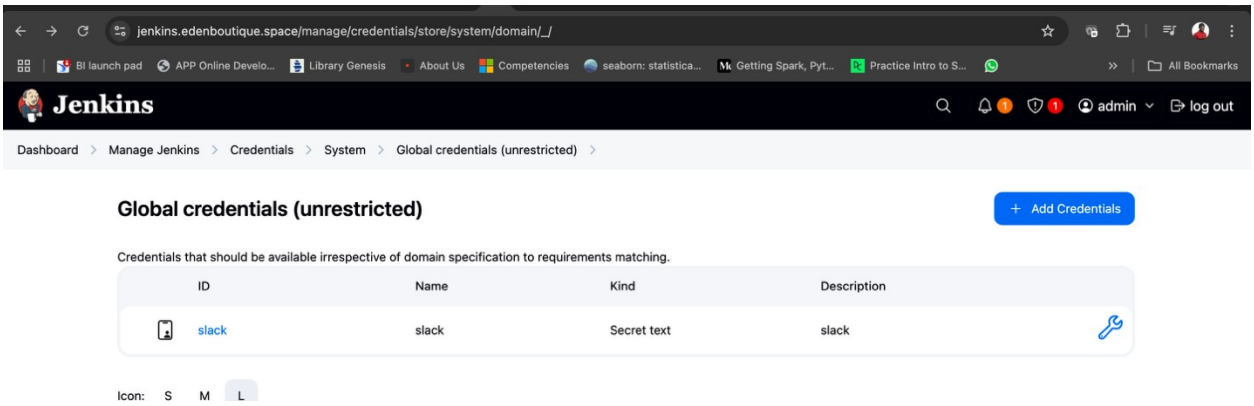


Scroll down to copy the token Credentials

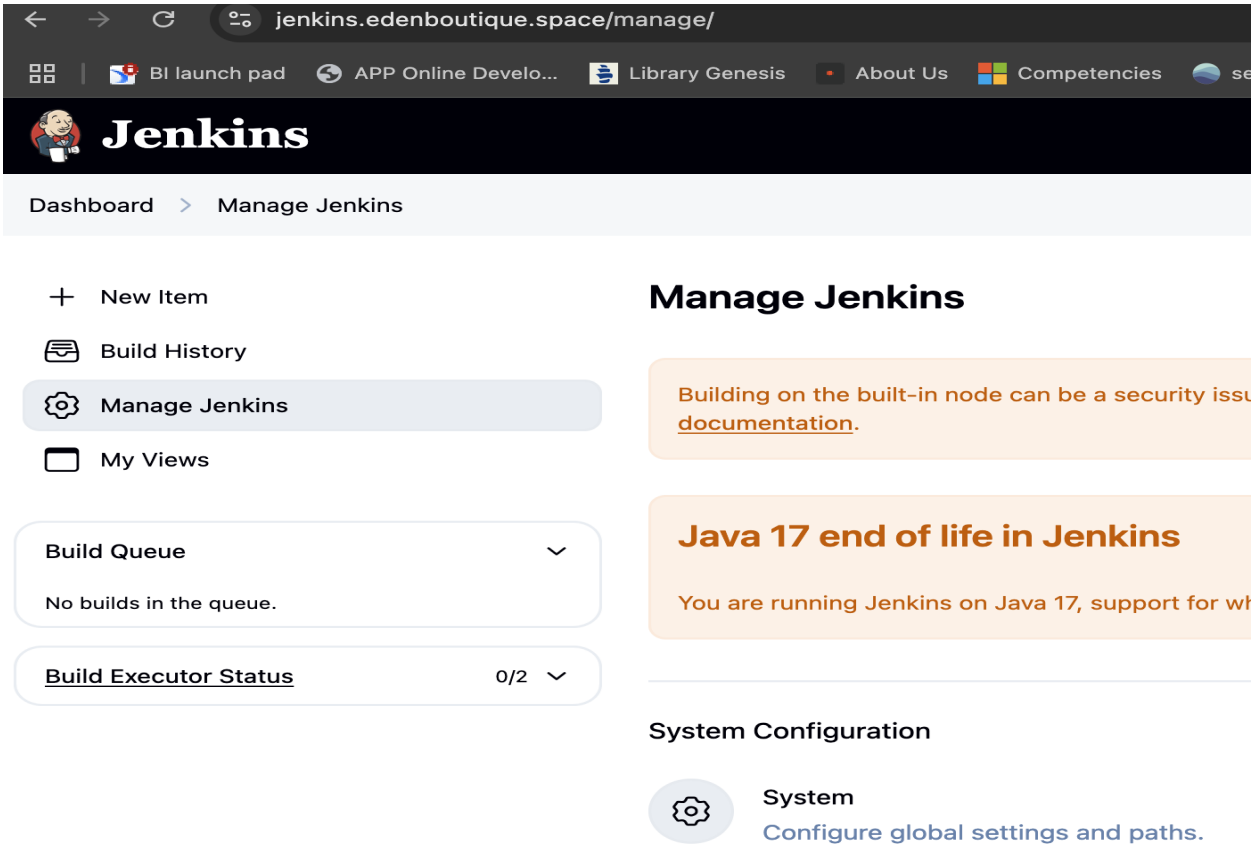


Paste the token in the “secret tab” and give your slack credential a name,later, click on. create

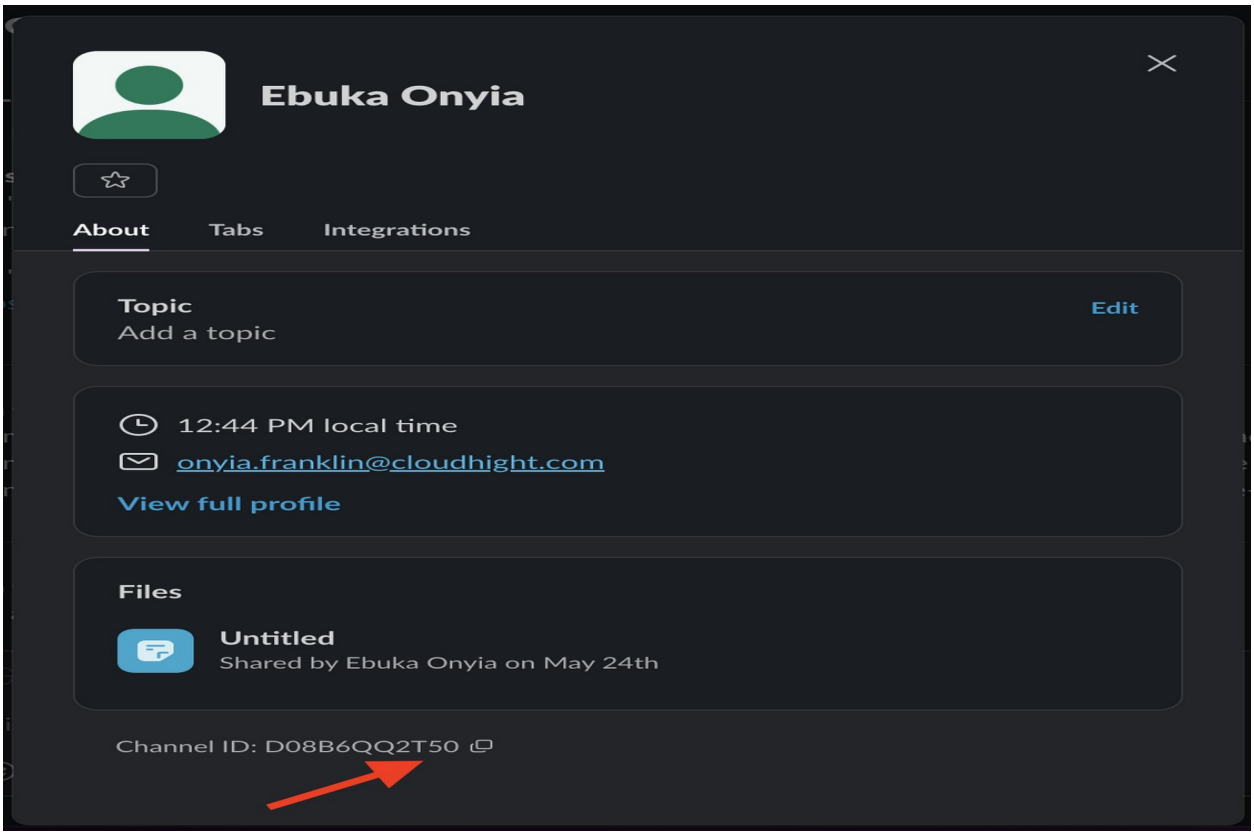




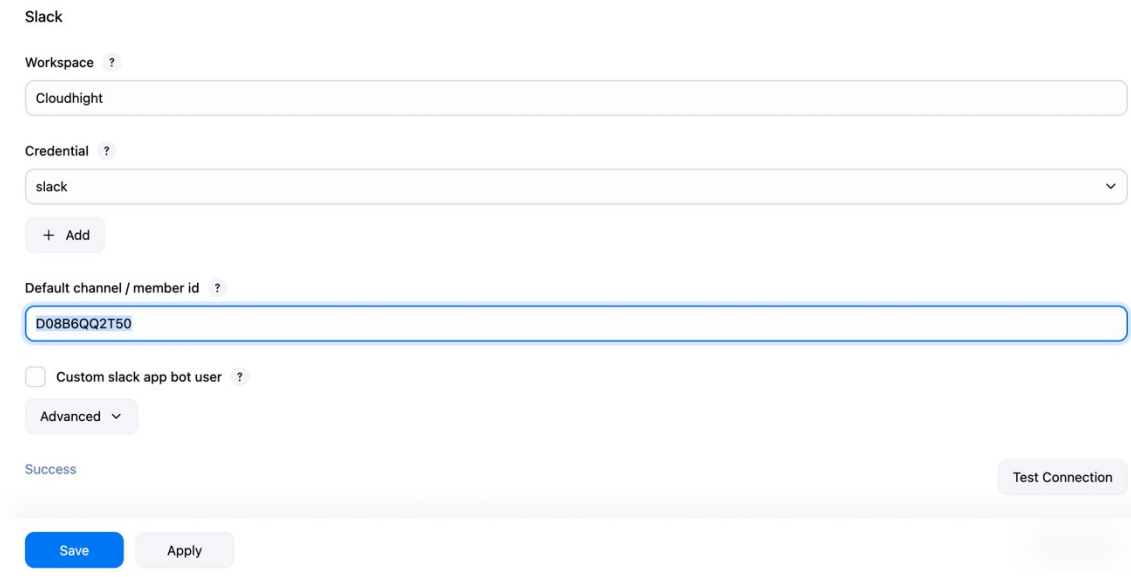
Next is to set the slack as a tool. Navigate to manage Jenkins and click on tools.

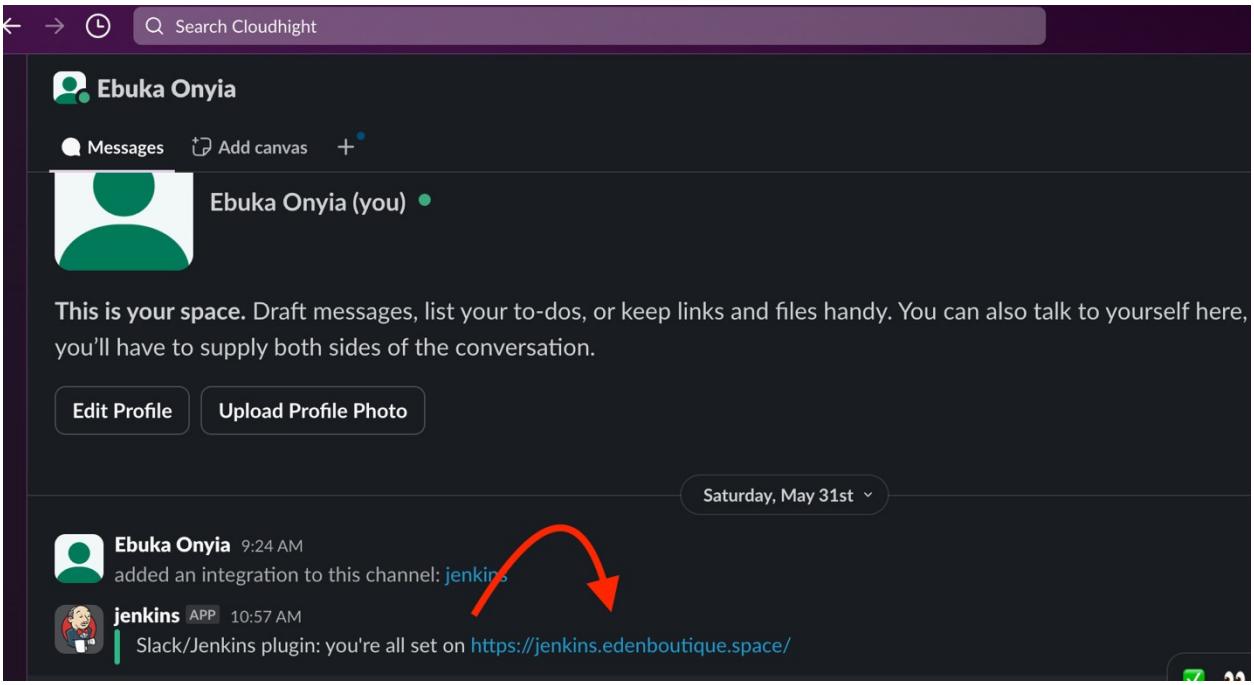


scroll down to slack. Add workspace where you like the notification to be and test connection. Because we set our personal channel for the notification, visit your personal channel and copy the channel ID before adding it to the channel id space on Jenkins.

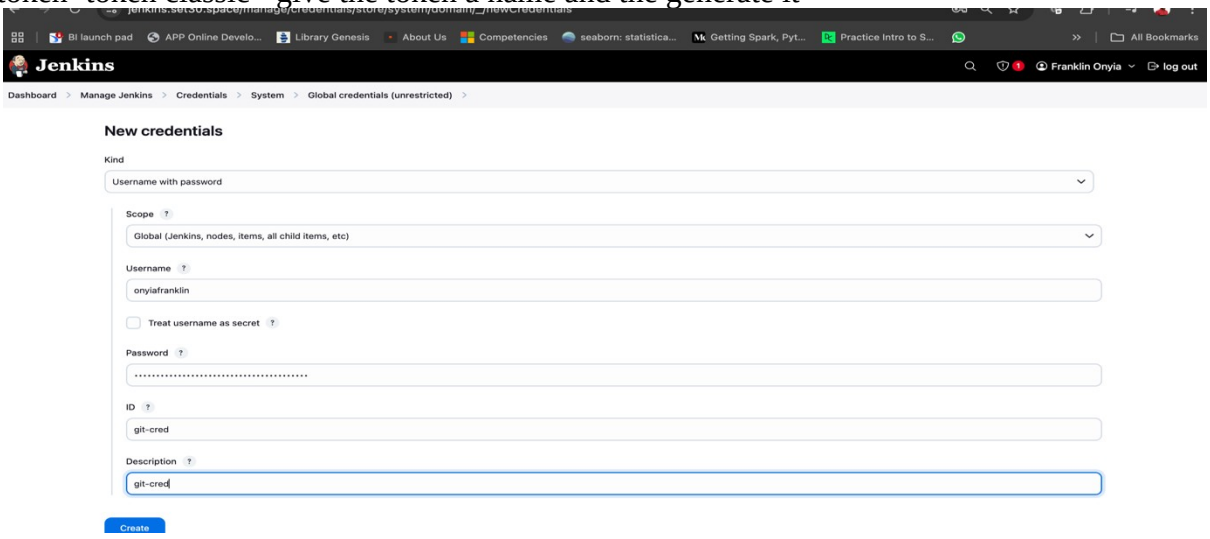


Click on test connection. You should get notification on your slack.
click on save and apply.

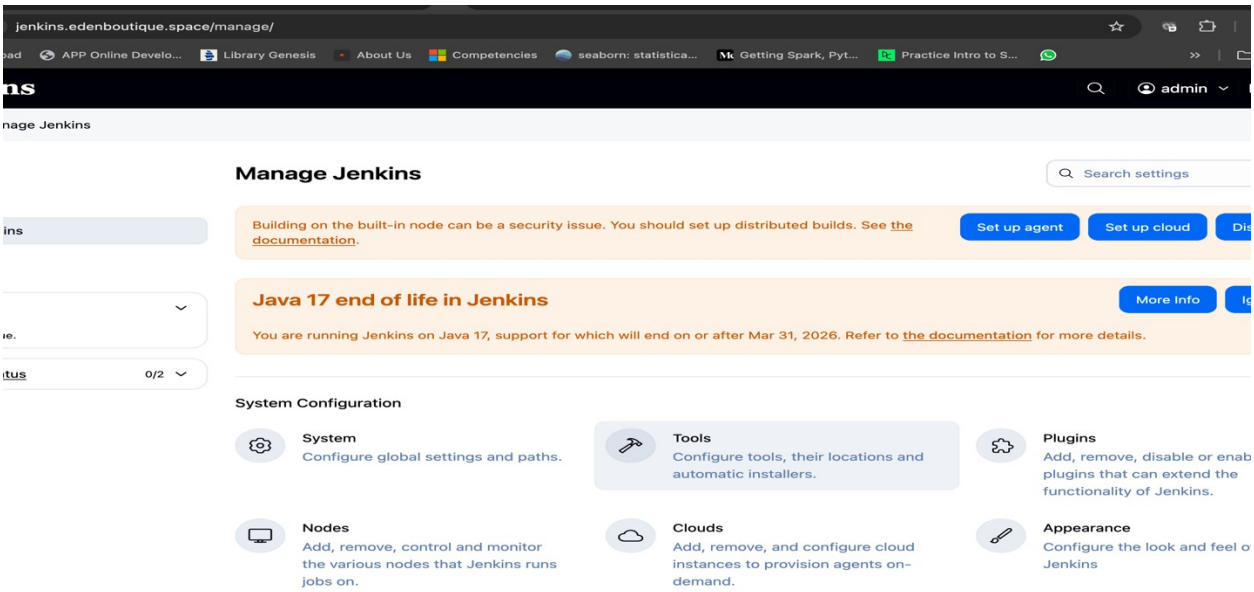




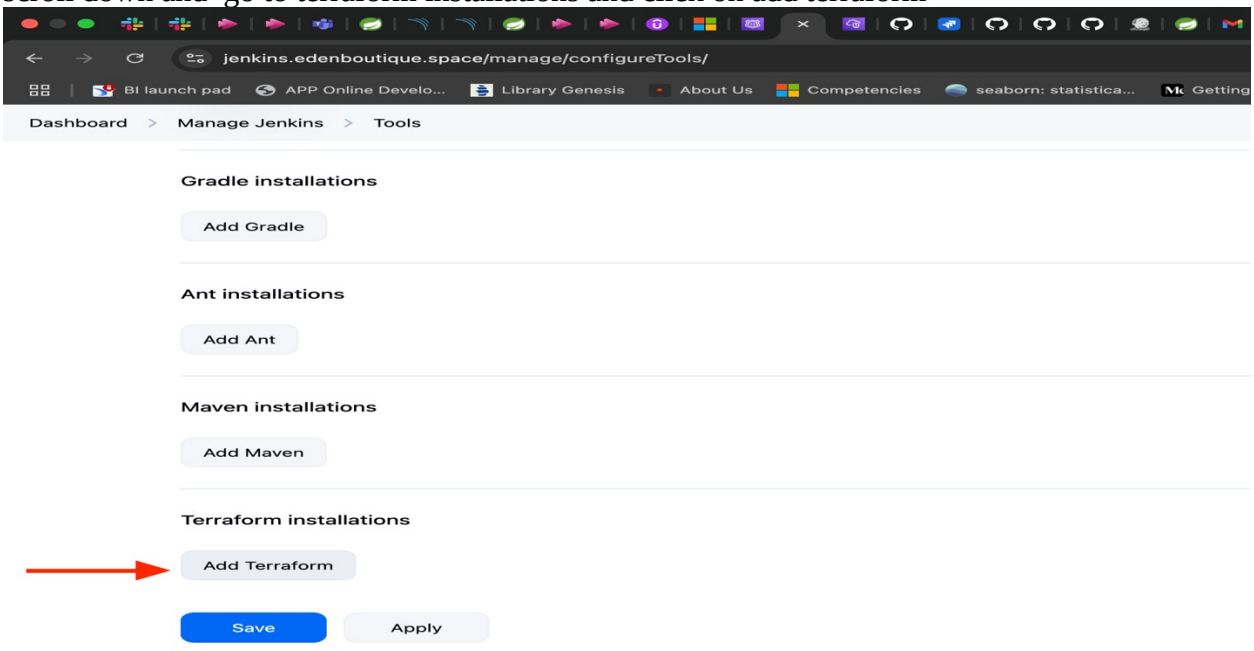
Now lets add the github credentials
Select username with password
enter the username of your github
Generate a classic token from your github account and enter the token in the password.
Do this by visiting your github account and Navigate to settings> developers settings>personal access token>token classic> give the token a name and the generate it



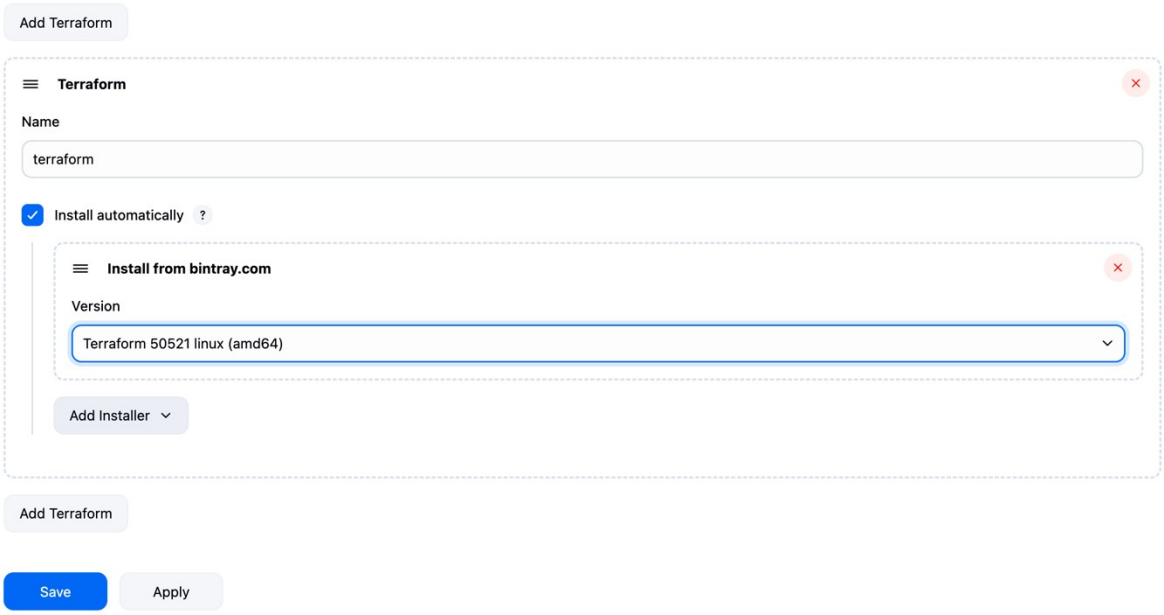
next , g back to tools on manage Jenkins , we want to install terraform tool



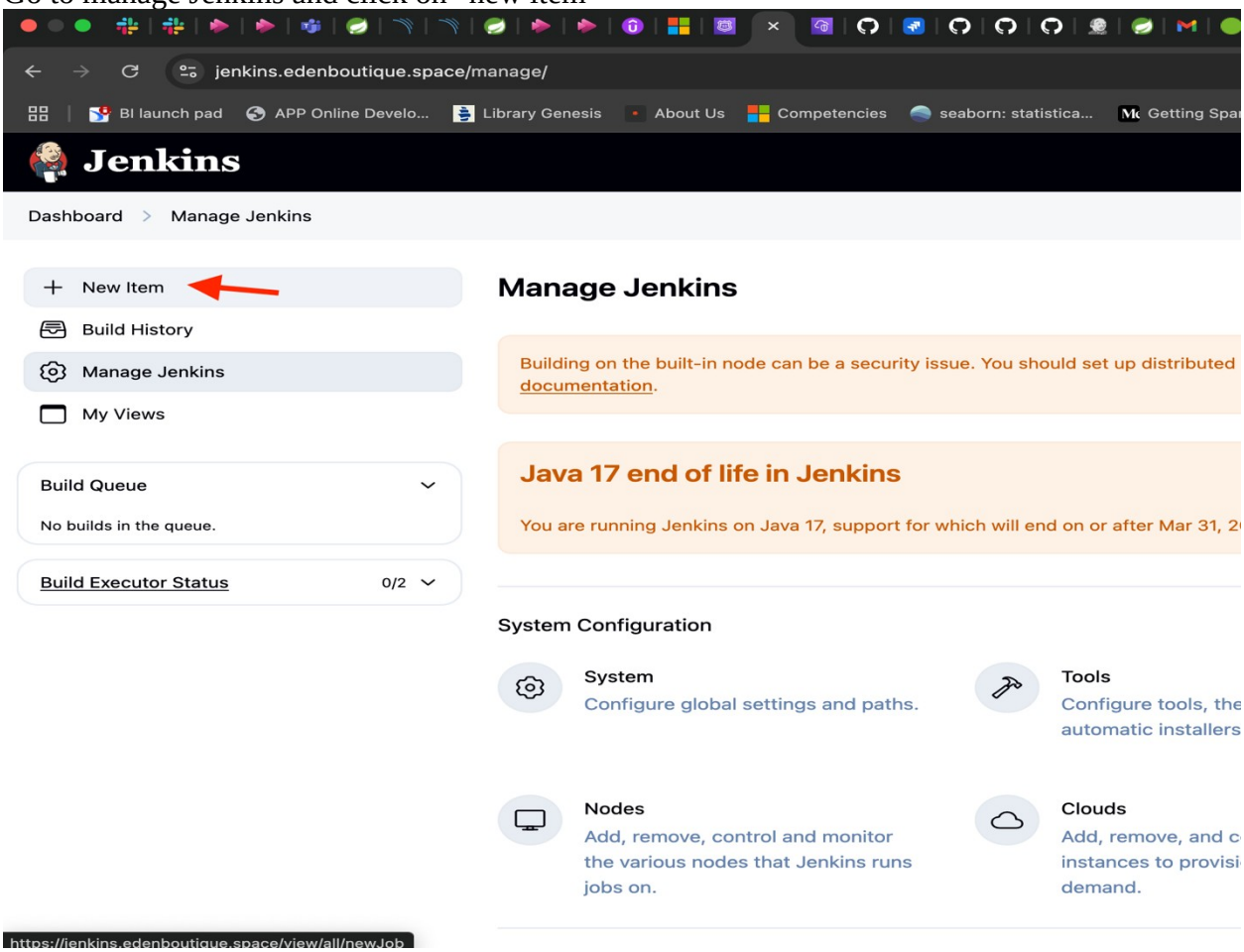
scroll down and go to terraform installations and click on add terraform



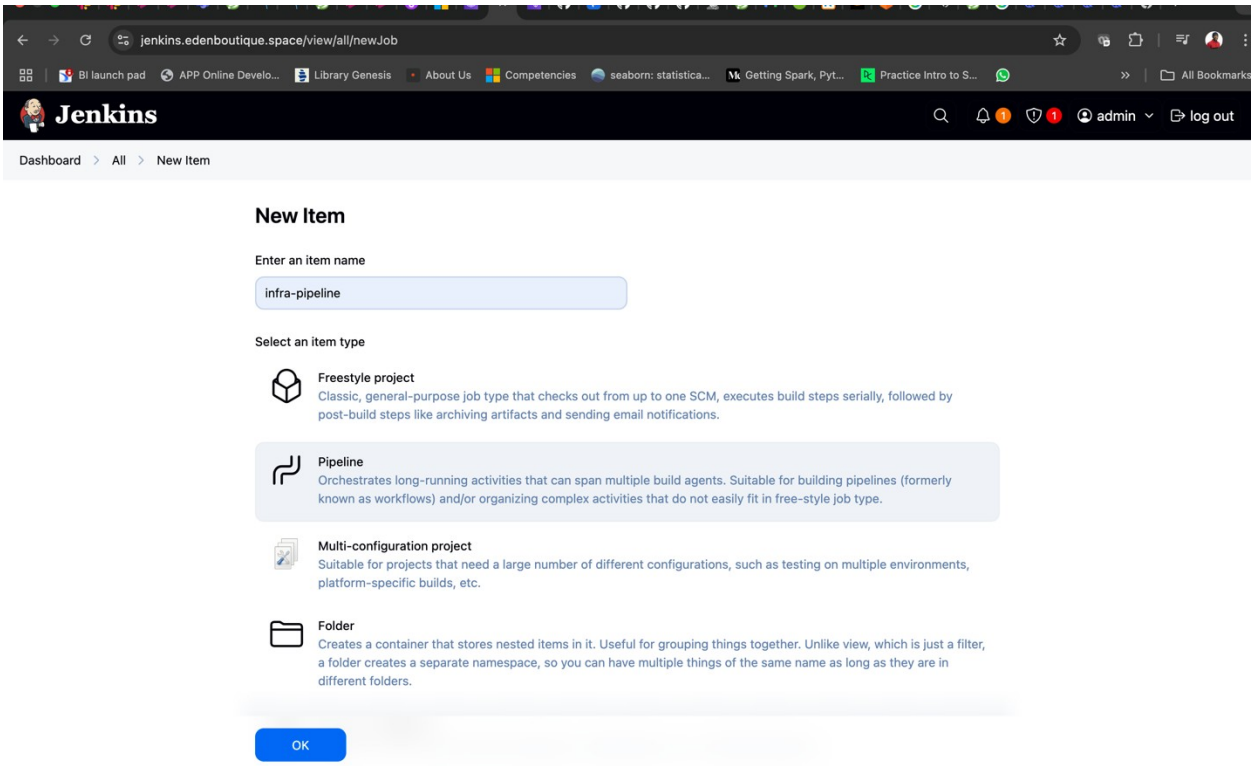
Select install automatically and select linux (amd64)



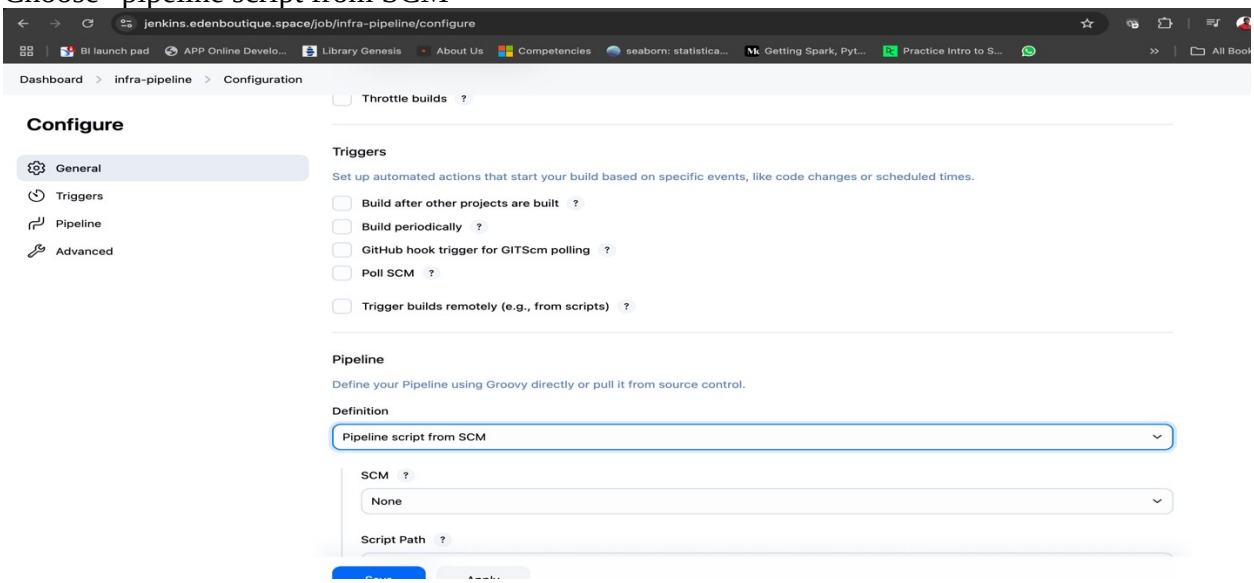
Now we can create our infrastructure pipeline
Go to manage Jenkins and click on “new item”



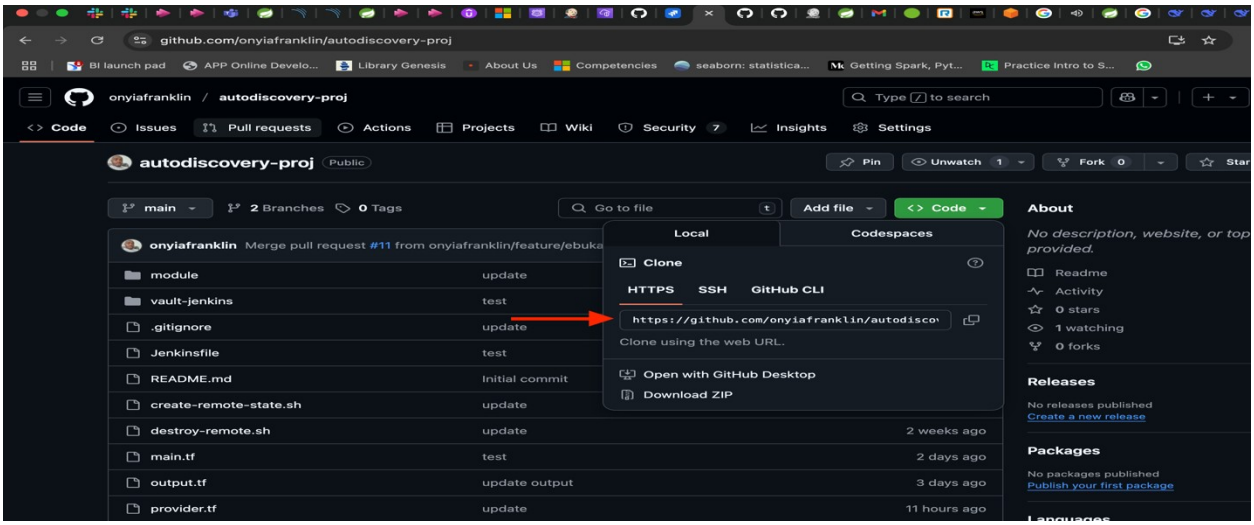
Give the pipeline a name and click ok



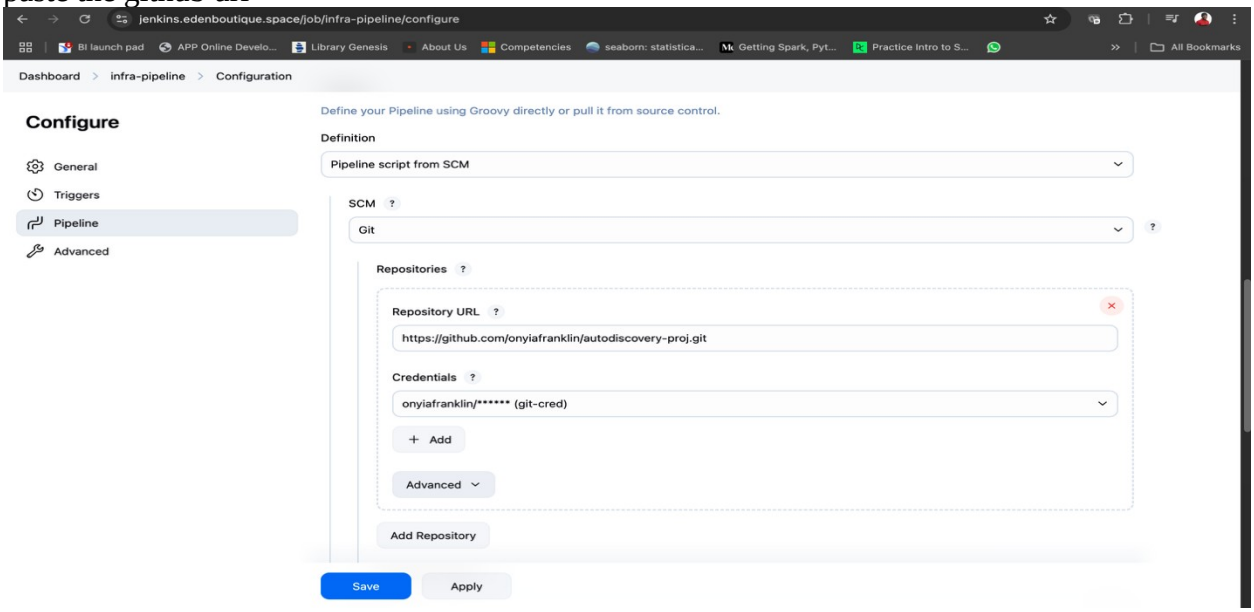
Choose “pipeline script from SCM”



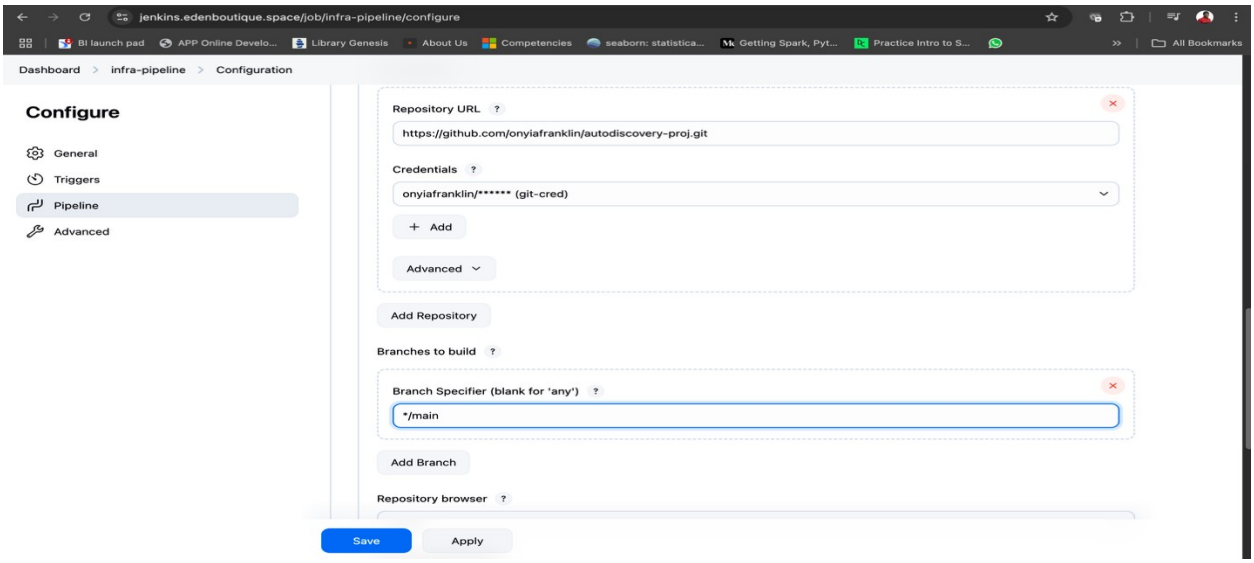
Copy your github url



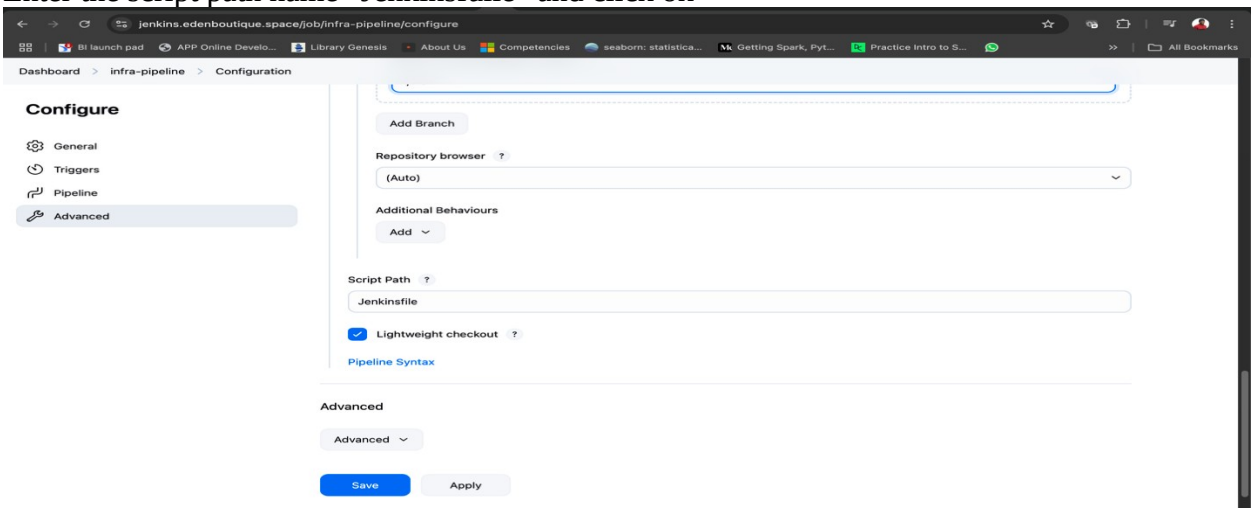
paste the github url



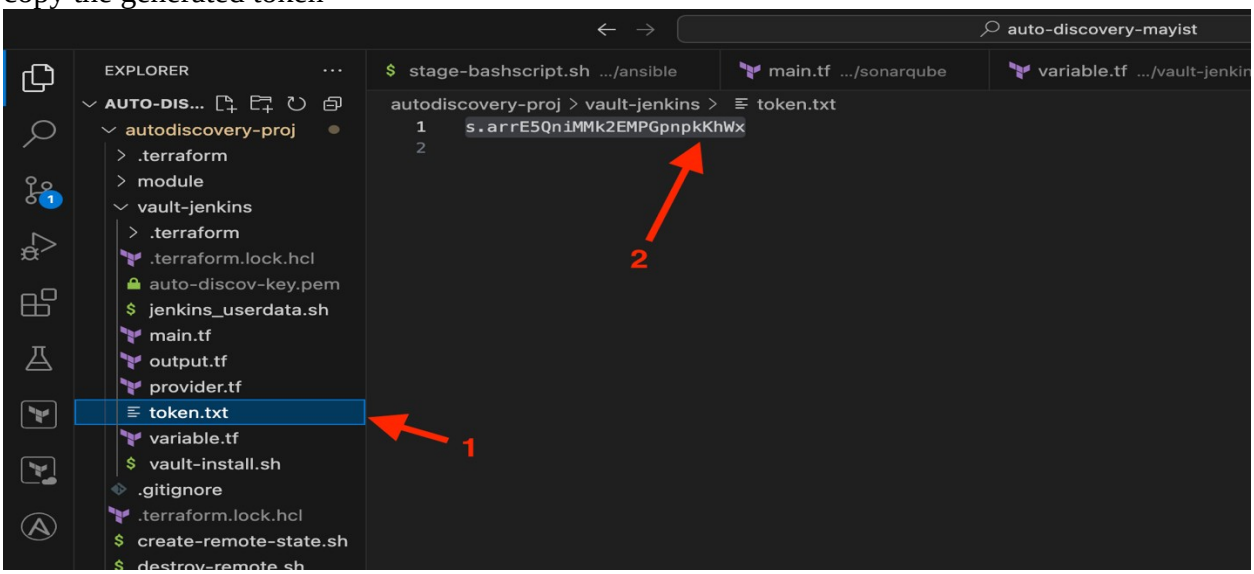
Select the main brach wher I pushed the code



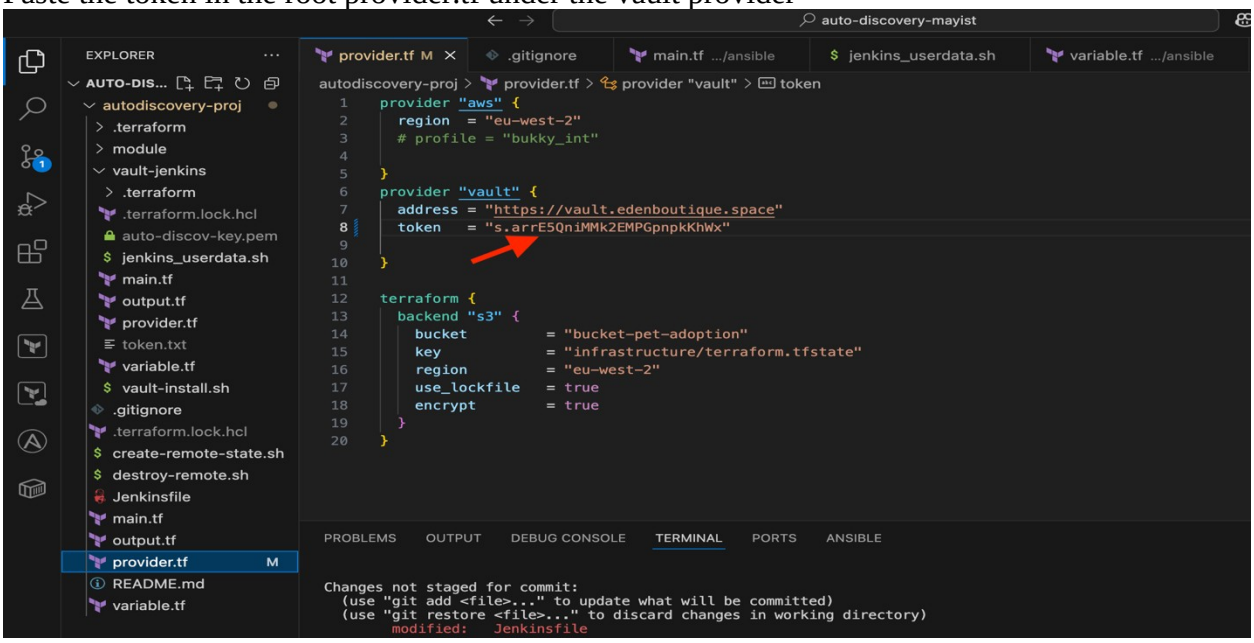
Enter the script path name “Jenkinsfuile” and click ok



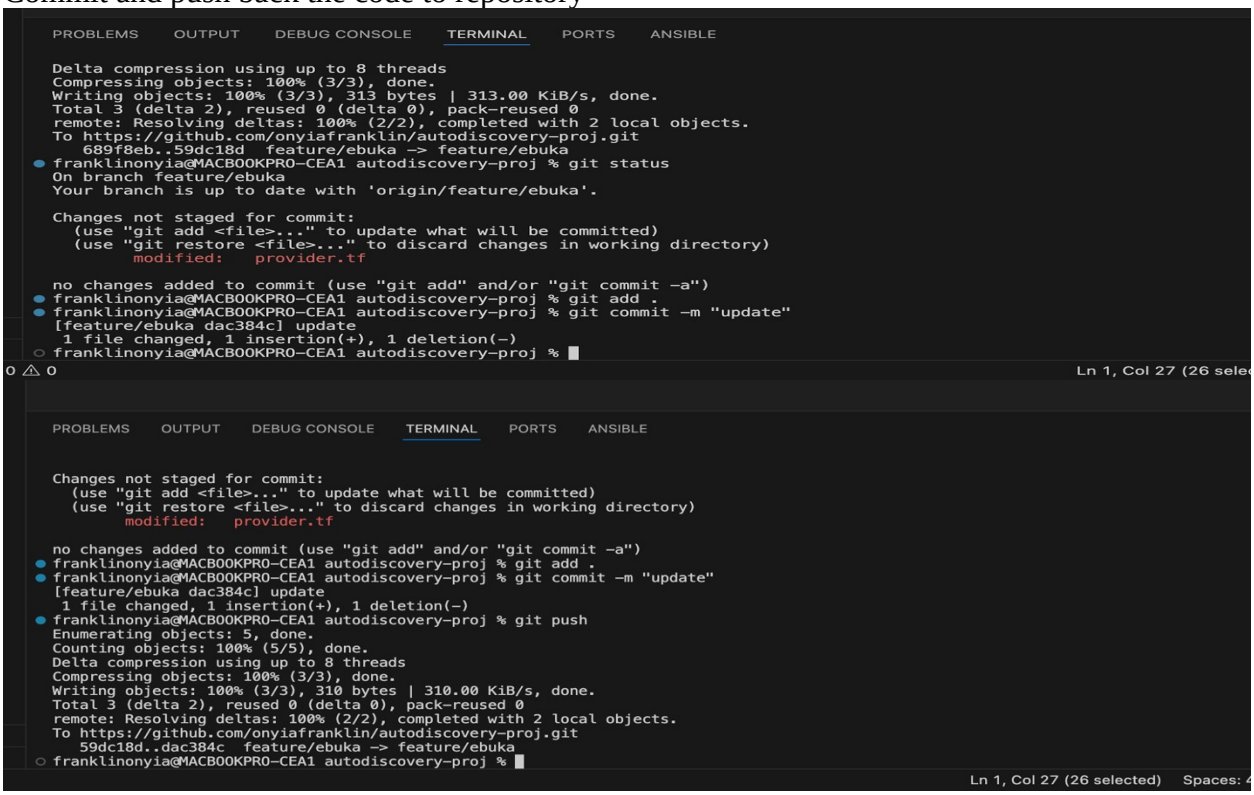
next is to update the vault token that would be used to unseal the vault
copy the generated token



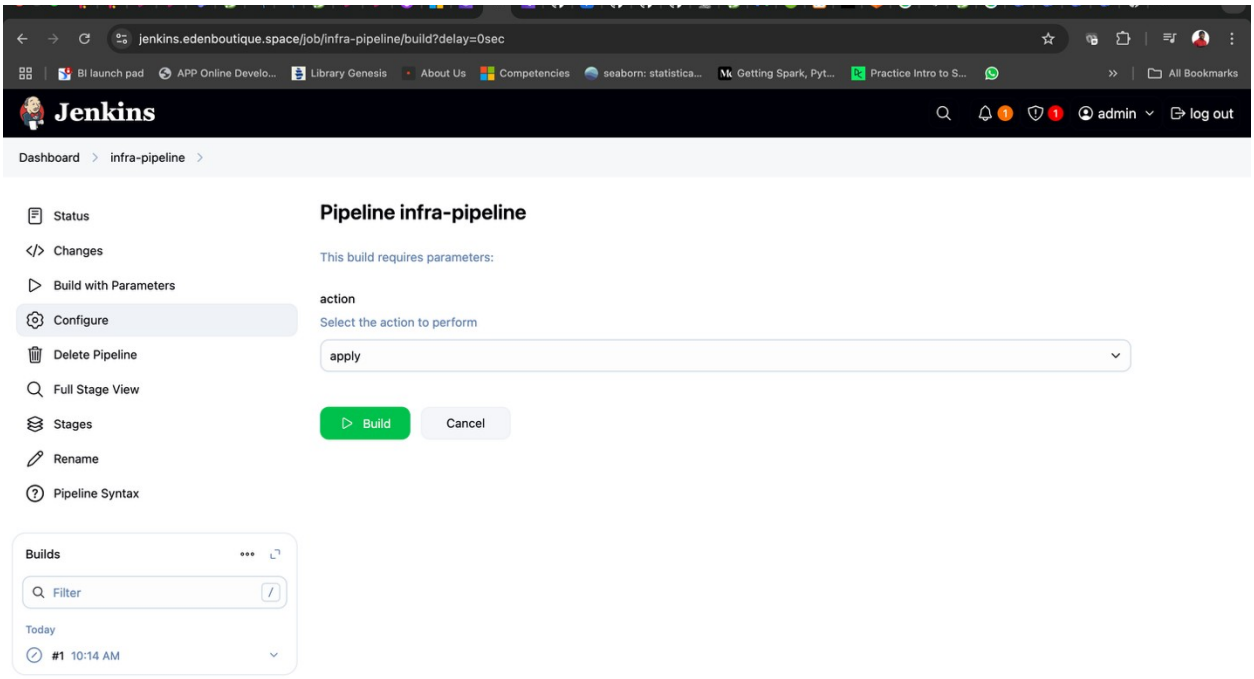
Paste the token in the root provider.tf under the vault provider



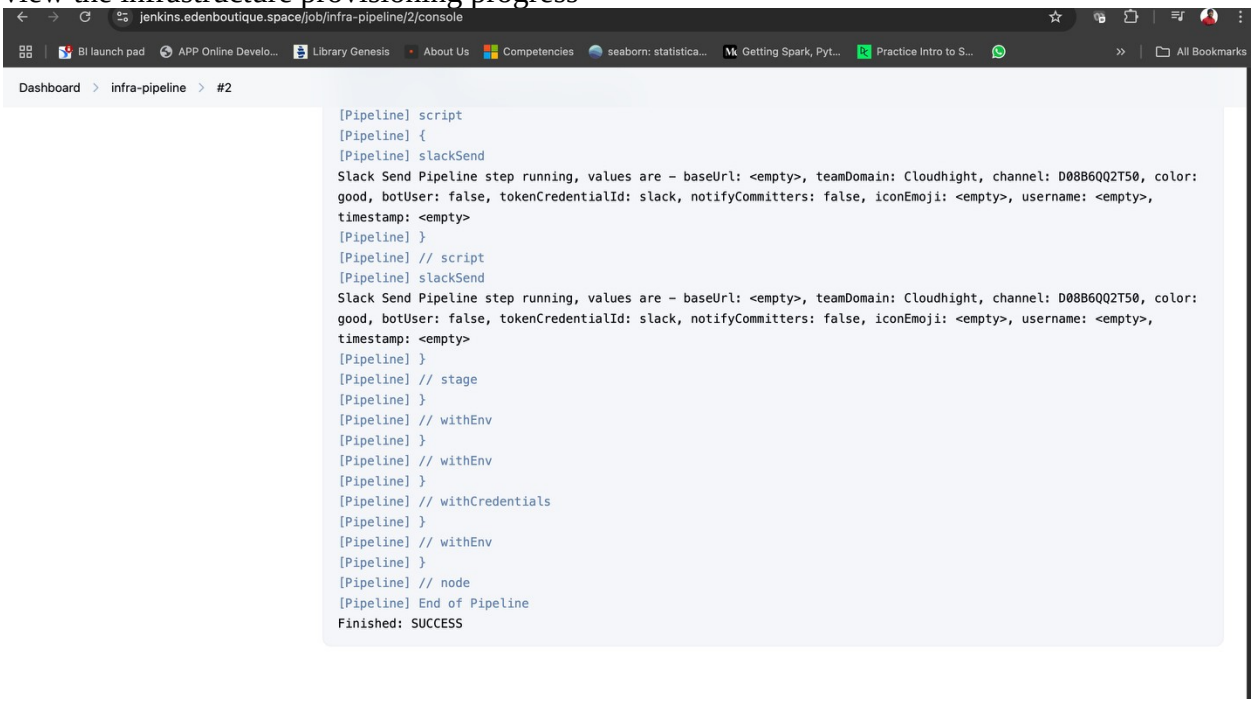
Commit and push back the code to repository



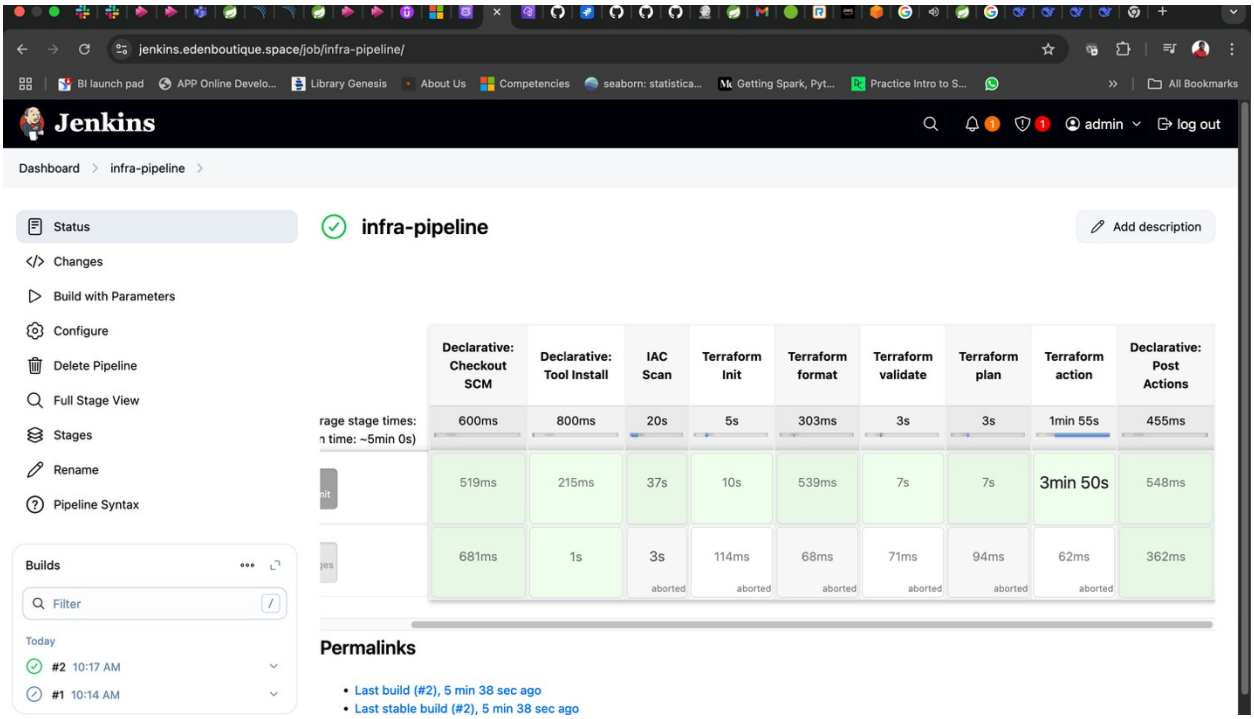
Go back to the Jenkins pipeline, refresh the url and click on “Build with parameters”
select “build”



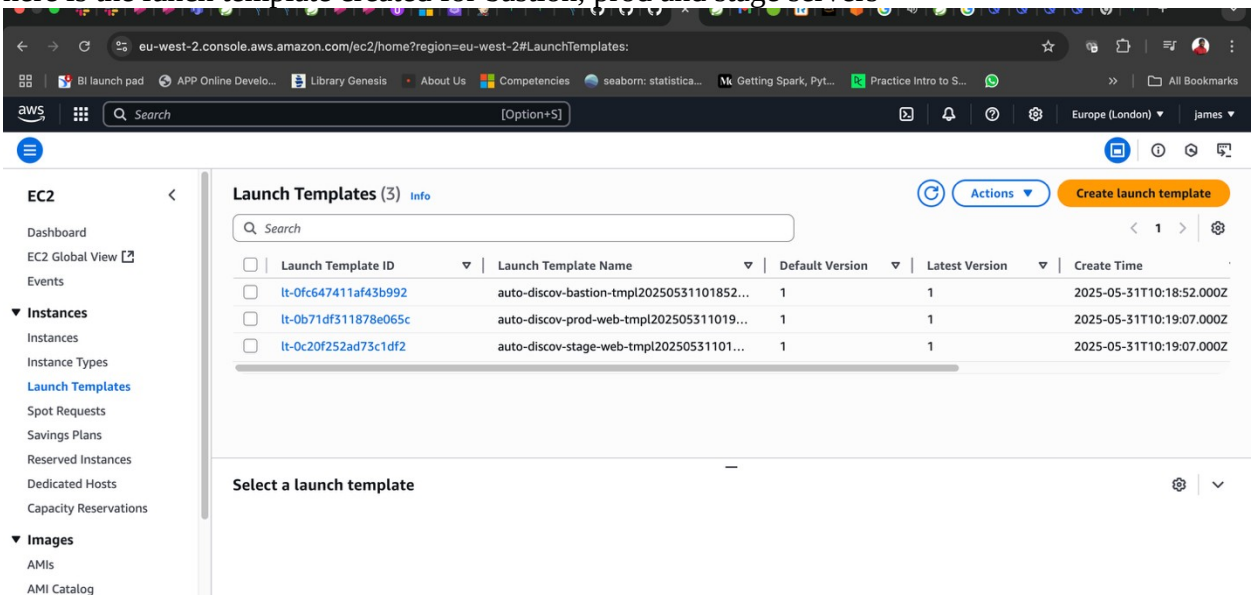
view the infrastructure provisioning progress



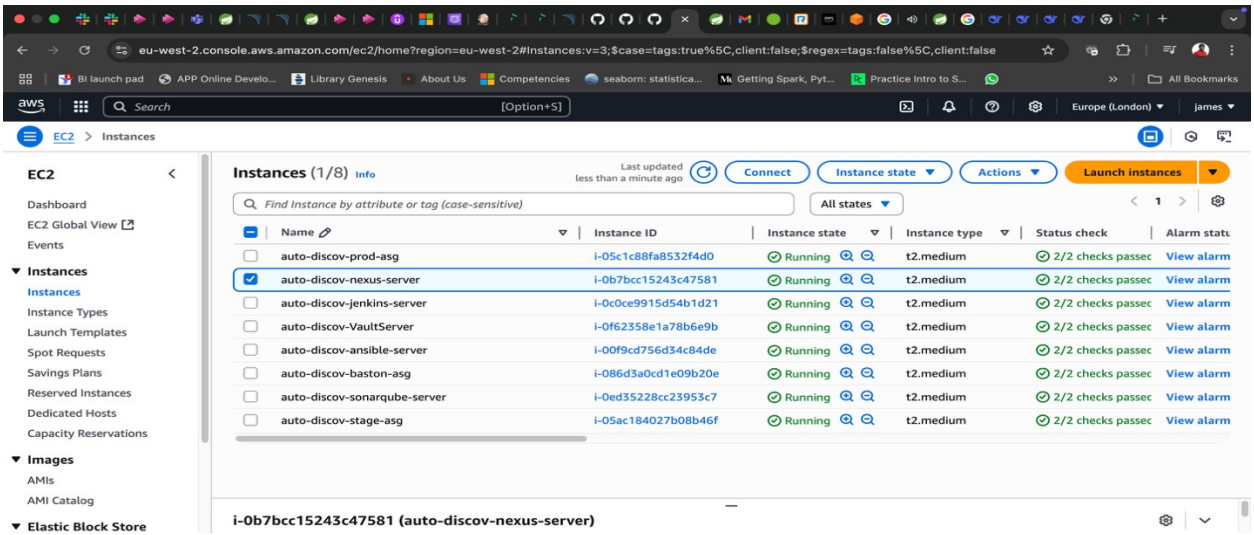
Infrastructure successfully deployed



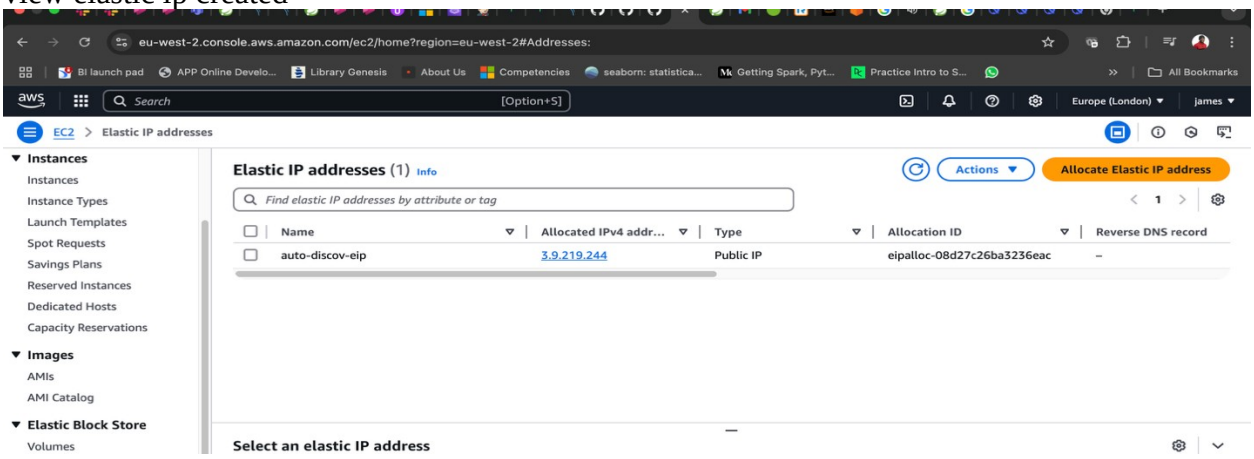
View some resources created using terraform on console
here is the lunch template created for bastion, prod and stage servers



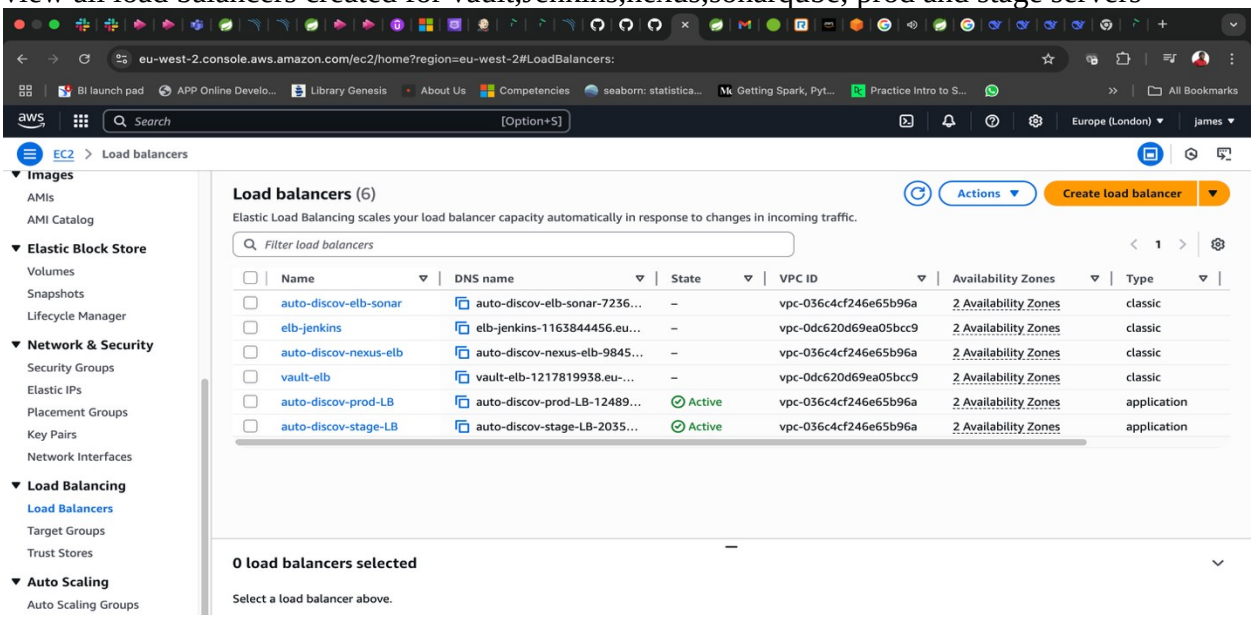
View all servers provisioned(prod,nexus,Jenkins,bastion,sonarqube, prod and stage all running)



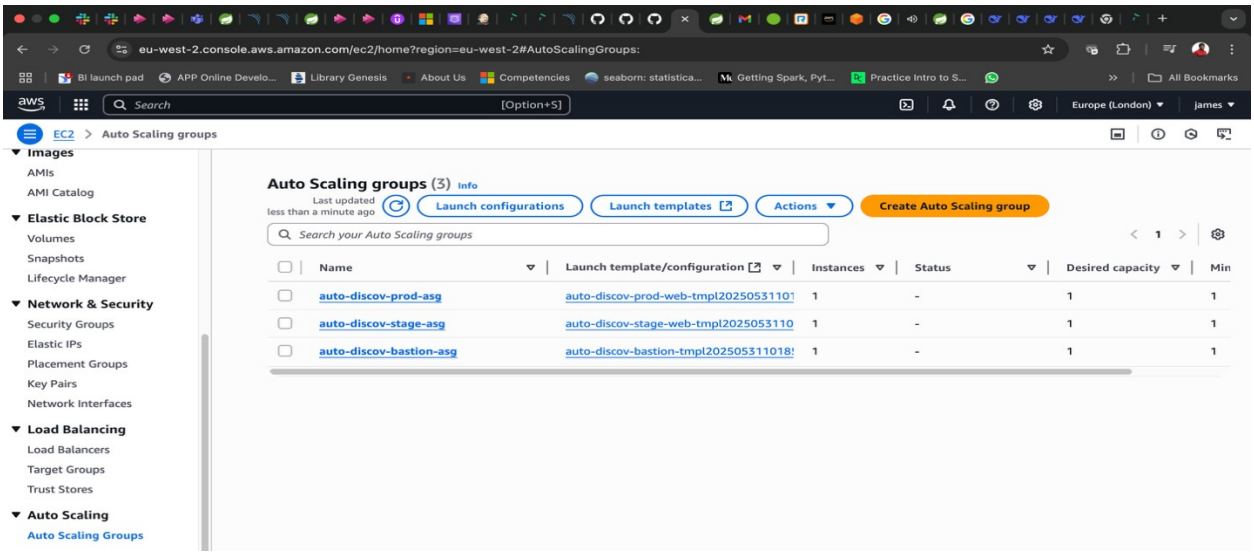
view elastic ip created



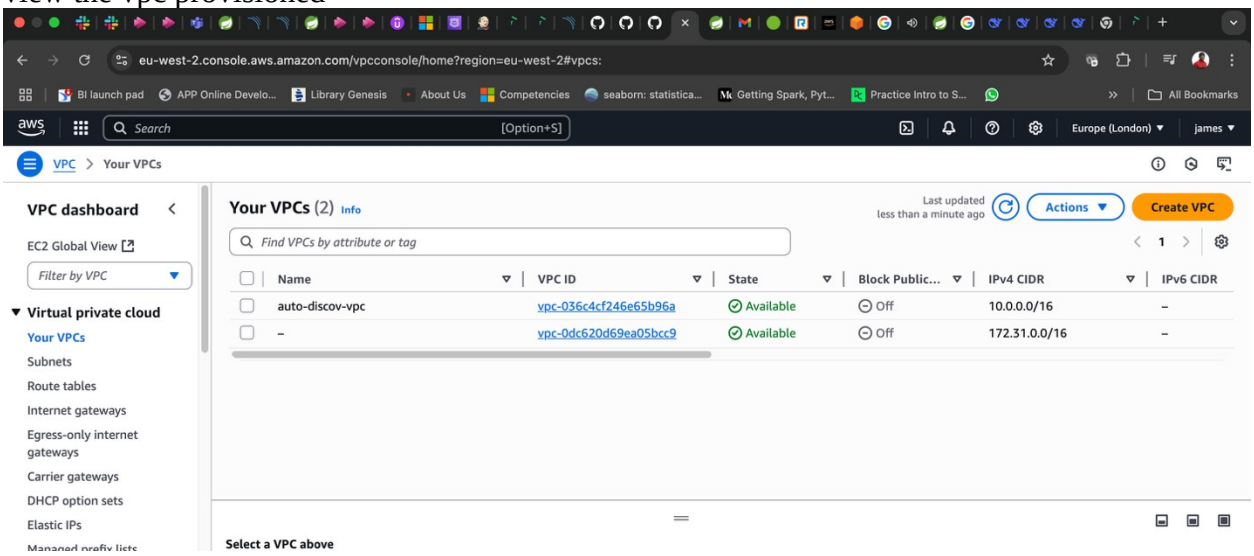
view all load balancers created for vault,Jenkins,nexus,sonarqube, prod and stage servers



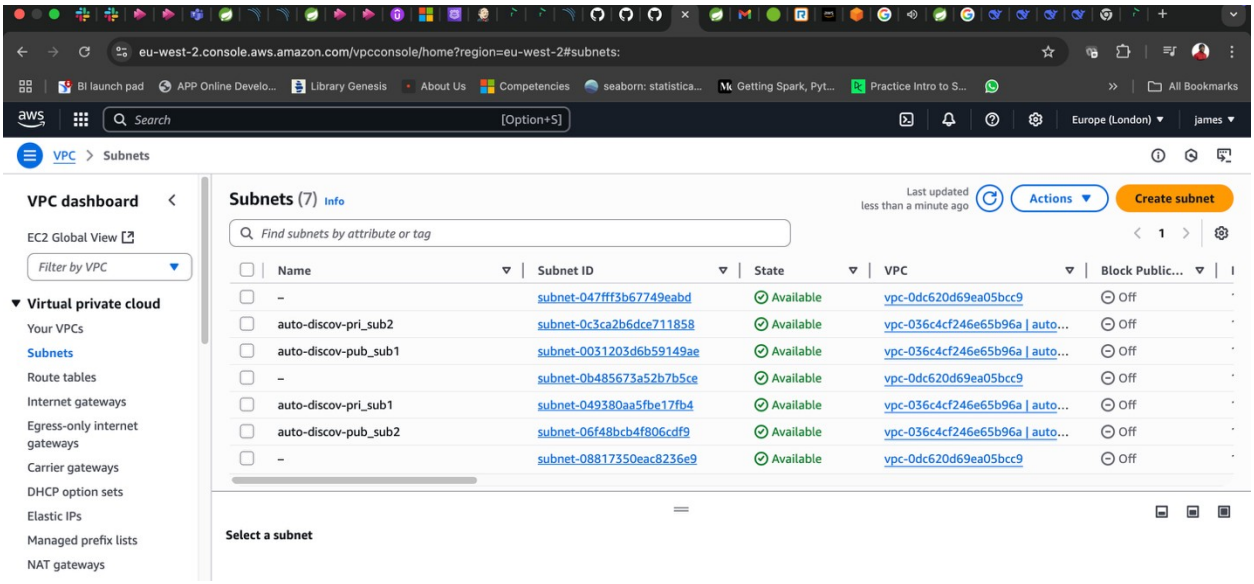
see auto scaling groups created for bastion, stage and prod servers



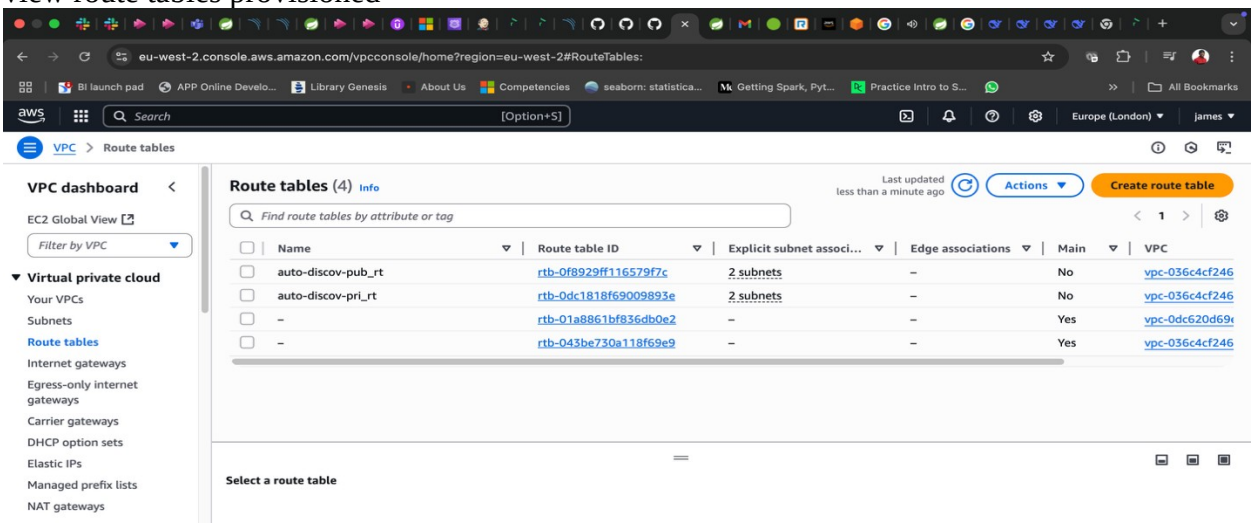
view the vpc provisioned



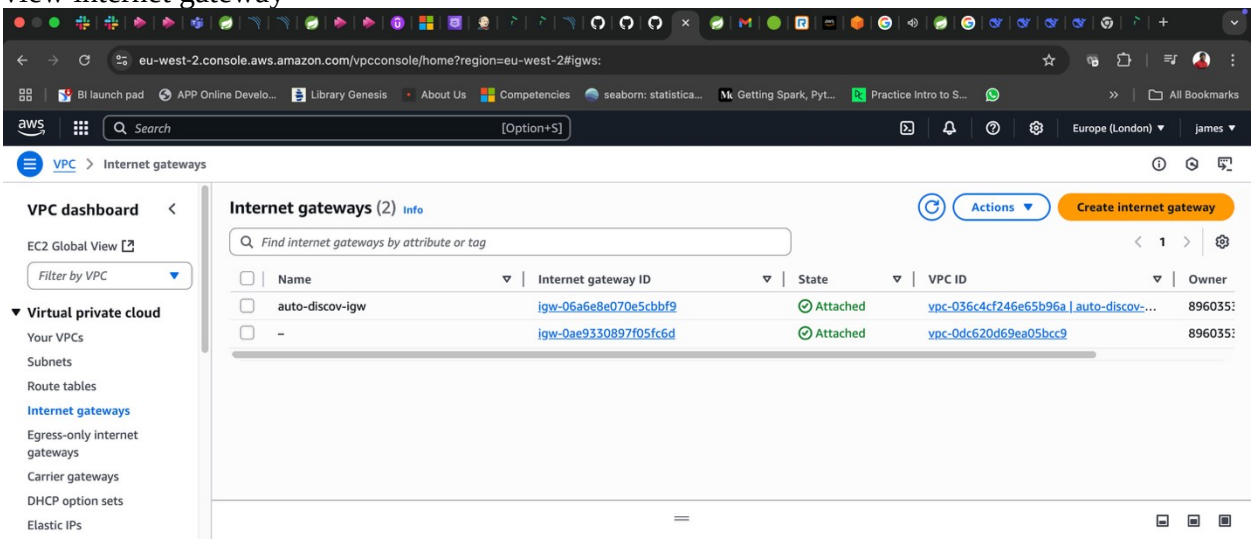
view the subnets provisioned



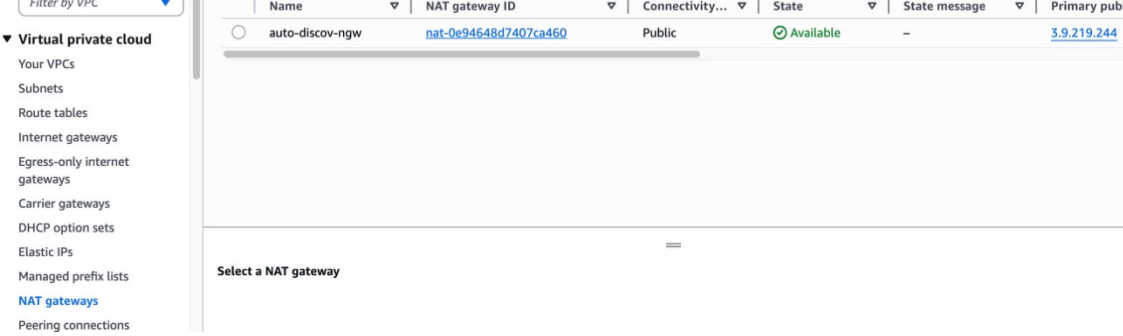
view route tables provisioned



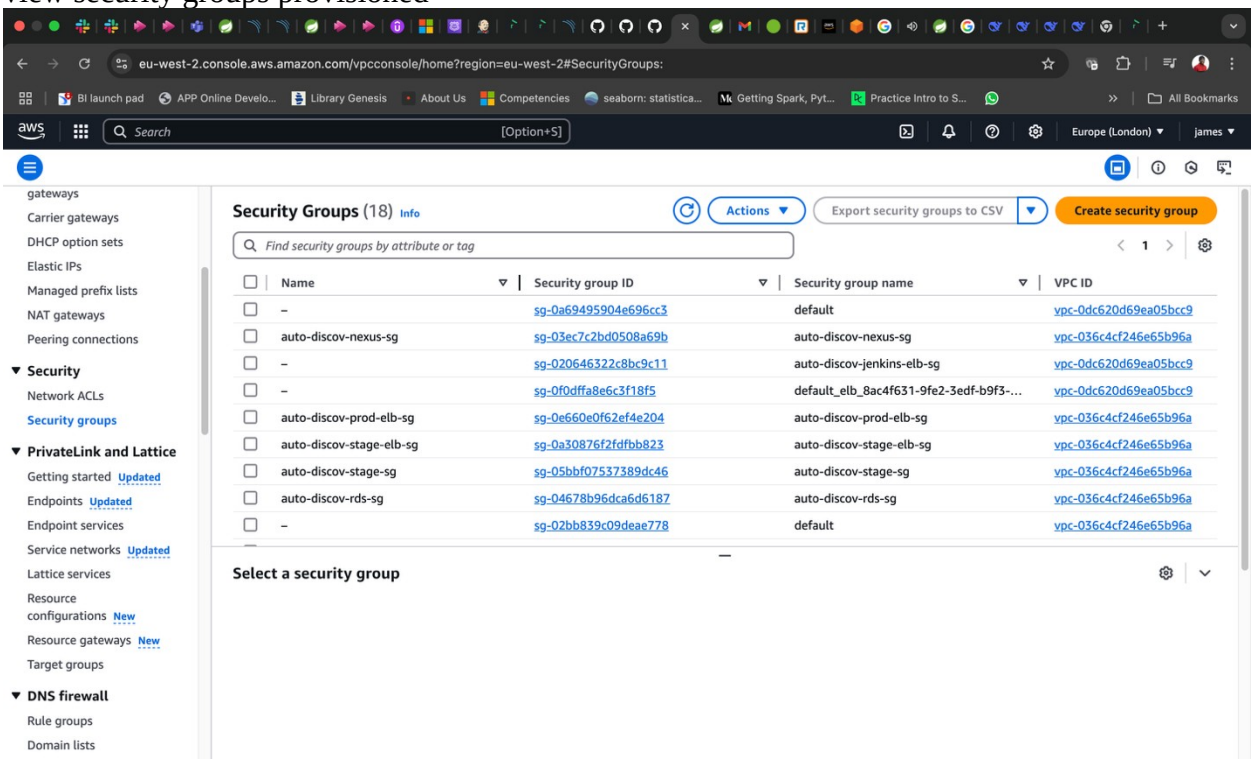
view internet gateway



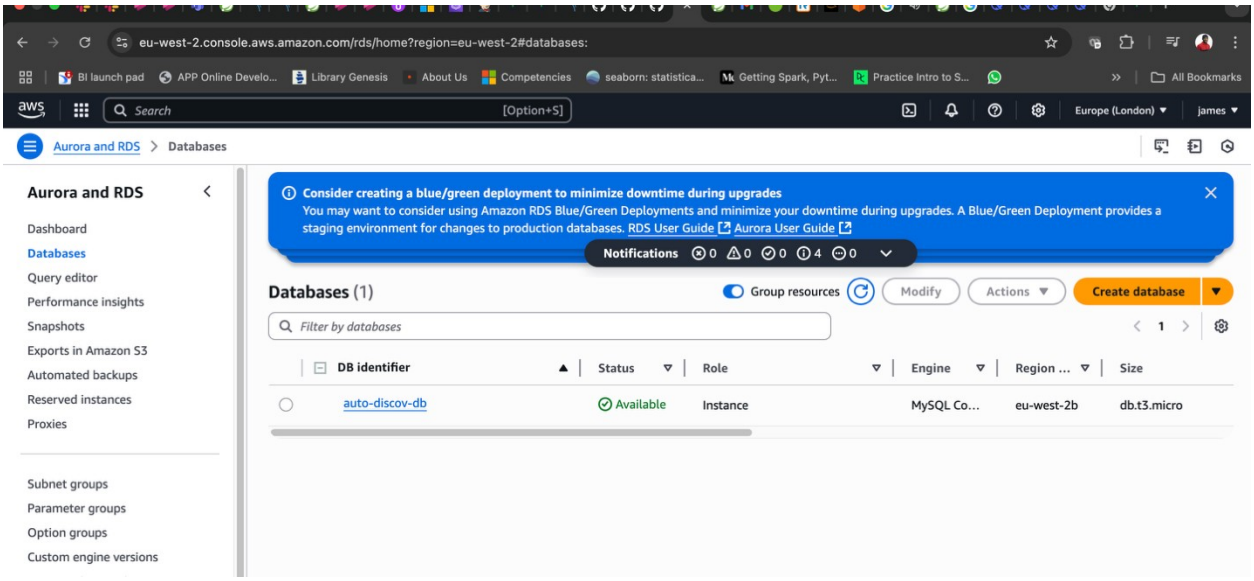
view natgateway



view security groups provisioned



view database provisioned



view scripts copied from the s3 bucket to ansible server by ssh into the ansible server

```
You can learn more about how to register your system
using rhc at https://red.ht/registration
Last login: Sat May 31 17:05:02 2025 from 10.0.1.198
[ec2-user@ansible-server ~]$ ls /etc/ansible
ansible.cfg           deployment.yml  prod-bashscript.sh  prod.lists  stage-bashscript.sh  stage.lists
ansible_vars_file.yml  hosts          prod_hosts          roles       stage_hosts
[ec2-user@ansible-server ~]$ cat /etc/ansible/stage_hosts
[webservers]
10.0.4.220 ansible_user=ec2-user ansible_ssh_private_key_file=/home/ec2-user/.ssh/id_rsa
[ec2-user@ansible-server ~]$
```

check to see the content that would be copied inside the ansible host

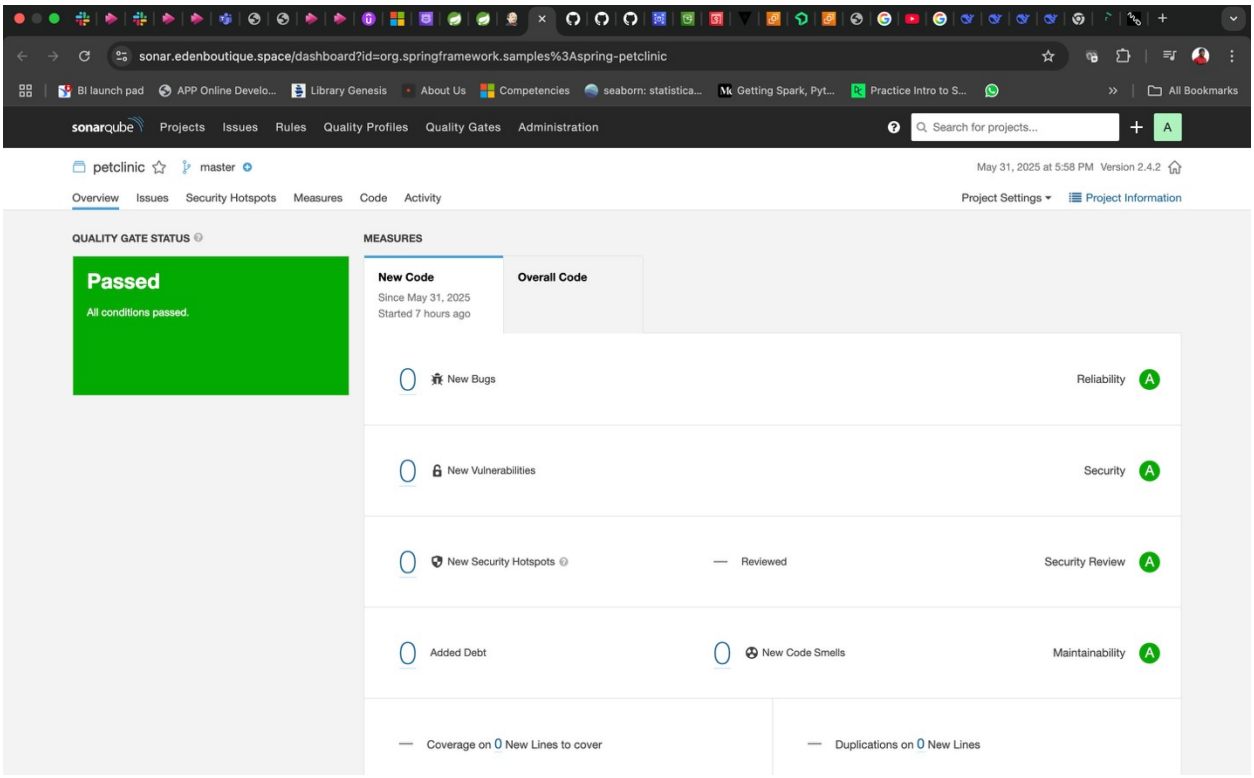
```
[ec2-user@ansible-server ~]$ cat /etc/ansible/prod_hosts
[webservers]
10.0.3.247 ansible_user=ec2-user ansible_ssh_private_key_file=/home/ec2-user/.ssh/id_rsa
[ec2-user@ansible-server ~]$
```

Ln 1, Col 1

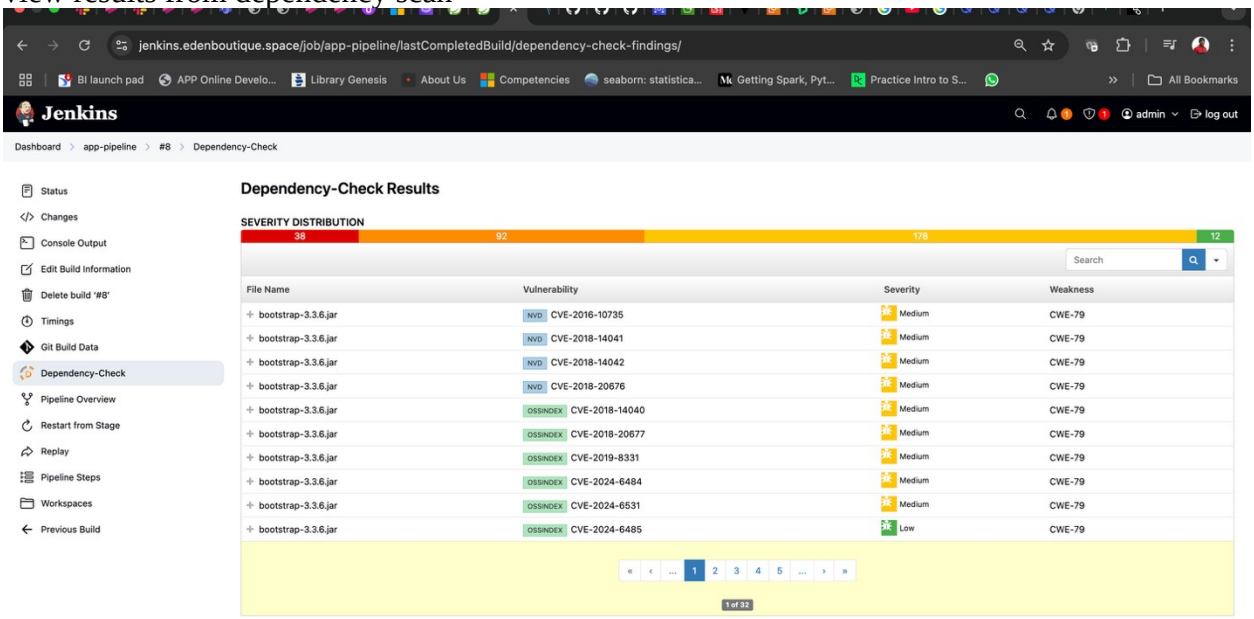
do same for stage servers

```
10.0.3.247
[ec2-user@ansible-server ~]$ cat /etc/ansible/stage.lists
10.0.4.220
[ec2-user@ansible-server ~]$
```

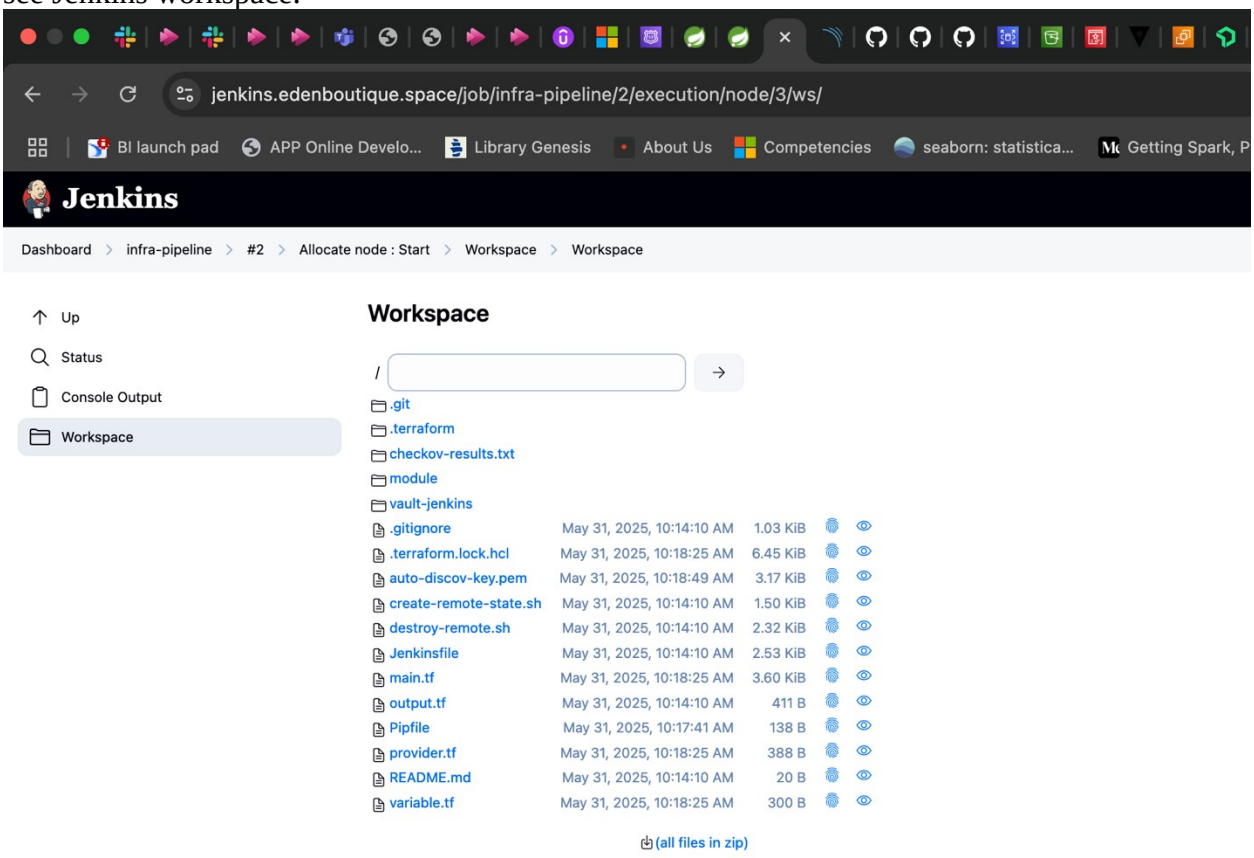
View results from quality gate scan



view results from dependency scan



see Jenkins workspace.`



Next is to deploy our application on already deployed infrastructure
to do that, we need to deploy our application on a different pipeline.
lets first of all write our Jenkins and docker file that our Jenkins pipeline would run.

I. APPLICATION JENKINSFILE EXPLANATION

The application pipeline starts with declaration of pipeline that encloses the whole pipeline stages its environment and agent to run the pipeline. Because we have not specified any agent to run the pipeline, it uses any available server , which is the jenkins server to run this pipeline.
next, we declared some environment variable that would be used in this pipeline

nexus user , nexus password an d nexus-docker-repo are credentials that jenkins would use to login to the docker nexus repo to run the docker image.

here is the explanation of the stages of the pipline
Here's a detailed breakdown of your Jenkins pipeline stages with clear headings for each stage:

1. Code Analysis Stage

Tool: SonarQube
Purpose: Performs static code analysis to identify bugs, vulnerabilities, and code quality issues
Process:

- Executes `mvn sonar:sonar` command to scan the Maven project
- Integrates with pre-configured SonarQube server ('sonarqube')
- Generates detailed reports on code metrics (coverage, duplications, security issues)

2. Quality Gate

Tool: SonarQube Quality Gate
Purpose: Ensures code meets quality standards before proceeding
Process:

- Waits for SonarQube analysis results (timeout: 2 minutes)
- Automatically aborts pipeline if quality thresholds aren't met

3. Dependency Check

Tool: OWASP Dependency-Check + NVD API
Purpose: Identifies vulnerable dependencies in the project
Process:

- Scans project dependencies using NVD API key for vulnerability data .NB:NVD-key help the dependency scan process to be faster
- Generates report (`dependency-check-report.xml`)
- Explicitly disables Node/Yarn audits (focuses on Java dependencies)

4. Build Artefacts

Tool: Maven
Purpose: Compiles application into deployable package
Process:

- Runs `mvn clean package` with tests and checkstyle skipped
  on this stage, this is the life cycle of what happens

. `clean` Phase (from the Clean Lifecycle)

- Deletes the `target/` directory (if it exists), removing all previously compiled files and artifacts.

. `validate` (Default Lifecycle)

- Checks if the project structure is correct (e.g., `pom.xml` exists).

. `compile`

- Compiles the main Java source code (`src/main/java`) into bytecode (stored in `target/classes`).

. `test-compile`

- Compiles test source code (`src/test/java`) into bytecode (stored in `target/test-classes`).

. `test`

- Runs unit tests (using a testing framework like JUnit) from `src/test/java`.
- Fails the build if any test fails (unless skipped via `-DskipTests`).

`package`

- Packages the compiled code (`target/classes`) and resources (`src/main/resources`) into the specified packaging format (e.g., JAR, WAR, EAR).
    - For a JAR: Creates a `.jar` file in `target/`.
    - For a WAR: Creates a `.war` file for web applications.

- Produces `spring-petclinic-2.4.2.war` in target directory

5. Push Artifacts to Nexus-Repo

Tool: Nexus Artifact Uploader
Purpose: Stores WAR file in Nexus repository
Process:

- Uploads WAR file to `nexus.edenboutique.space` via HTTPS
- Uses credentials ('nexus-cred') for authentication
- Organizes artifact with:
    - Group ID: Petclinic
    - Artifact ID: spring-petclinic
    - Version: 1.0

6. Build Docker Image

Tool: Docker
Purpose: Creates containerized version of application
Process:

- Builds image from Dockerfile using `docker build`
- Tags image as `$NEXUS_REPO/petclinicapps`

7. Login to Nexus Repo

Tool: Docker CLI
Purpose: Authenticates to private Nexus Docker registry
Process:

- Uses stored credentials (`NEXUS_USER`, `NEXUS_PASSWORD`)
- Executes `docker login` command

8. Push Image to Nexus Repo

Tool: Docker CLI
Purpose: Stores Docker image in Nexus for deployment
Process:

- Pushes the built image to Nexus repository
- Makes image available for deployment stages

9. Trivy Image Scan

Tool: Trivy
Purpose: Security scans the Docker image
Process:

- Analyzes image for OS/library vulnerabilities
- Outputs results to `trivy.txt` file

10. Deploy to Stage

Tool: Ansible + SSH Agent
Purpose: Deploys application to staging environment
Process:

- Uses SSH agent with 'ansible-key' credentials
- Connects through bastion host (`BASTION_IP`) to executes Ansible playbook (`deployment.yml`) against stage hosts

11. Check Stage Website Availability

Tool: cURL + Slack
Purpose: Verifies successful staging deployment
Process:

- Waits 200 seconds for application to initialize
- Checks HTTP status of `stage.edenboutique.space`
- Sends Slack notification with results:
    - Success (HTTP 200): Green message
    - Failure: Red alert message

12. Request for Approval

Tool: Jenkins Input Step
Purpose: Manual approval before production deployment
Process:

- Pauses pipeline for up to 10 minutes
- Requires approval from 'admin' role

13. Deploy to Prod

Tool: Ansible + SSH Agent
Purpose: Deploys to production environment
Process:

- Similar to stage deployment
- Uses production inventory (`prod_hosts`)
- Same Ansible playbook but targets production servers

14. Check Prod Website Availability

Tool: cURL + Slack
Purpose: Validates production deployment
Process:

- Checks `www.edenboutique.space` after 200s warmup
- Sends Slack notification with status
- Differentiates between stage/prod in messages

```
pipeline {
    agent any
    environment {
        NEXUS_USER = credentials('nexus-username')
        NEXUS_PASSWORD = credentials('nexus-password')
```

```
        NEXUS_REPO = credentials('nexus-docker-repo')
        NVD_API_KEY= credentials('nvd-key')
        ANSIBLE_IP = credentials('ansible-ip')
        BASTION_IP = credentials('bastion-ip')
    }
    stages {
        stage('Code analisys stage') {
            steps {
                withSonarQubeEnv('sonarqube') {
                    sh 'mvn sonar:sonar'
                }
            }
        }
        stage('Quality gate') {
            steps {
                timeout(time: 2, unit: 'MINUTES') {
                    waitForQualityGate abortPipeline: true
                }
            }
        }
        stage('Dependency check') {
            steps {
                withCredentials([string(credentialsId: 'nvd-key', variable:
'NVD_API_KEY')]) {
                    dependencyCheck additionalArguments: "--scan ./ --
disableYarnAudit --disableNodeAudit --nvdApiKey $NVD_API_KEY",
                         odcInstallation: 'DP-Check'
                }
                dependencyCheckPublisher pattern: '**/dependency-check-
report.xml'
            }
        }
        stage('Build artefacts') {
            steps {
                sh 'mvn clean package -DskipTests -Dcheckstyle.skip'
            }
        }

        stage('Push artifacts to nexus-repo') {
            steps {
                nexusArtifactUploader artifacts: [[artifactId: 'spring-
petclinic',
                classifier: '',
                file: 'target/spring-petclinic-2.4.2.war',
                type: 'war']],
                credentialsId: 'nexus-cred',
                groupId: 'Petclinic',
                nexusUrl: 'nexus.edenboutique.space',
                nexusVersion: 'nexus3',
                protocol: 'https',
                repository: 'nexus-repo',
                version: '1.0'
            }
        }

        stage('Build Docker image') {
            steps {
```

```
                sh 'docker build -t $NEXUS_REPO/petclinicapps .'
            }
        }

        stage('Login to Nexus repo') {
            steps {
                sh 'docker login --username $NEXUS_USER --password
$NEXUS_PASSWORD $NEXUS_REPO'
            }
        }
        stage('Push image to Nexus repo') {
            steps {
                sh 'docker push $NEXUS_REPO/petclinicapps'
            }
        }
        stage('Trivy image scan') {
            steps {
                sh "trivy image $NEXUS_REPO/petclinicapps > trivy.txt"
            }
        }
        stage('Deploy to stage') {
            steps {
                sshagent(['ansible-key']) {
                    sh '''
                         ssh -t -t -o StrictHostKeyChecking=no -o
ProxyCommand="ssh -W %h:%p -o StrictHostKeyChecking=no ec2-user@$
{BASTION_IP}" ec2-user@${ANSIBLE_IP} "ansible-playbook -i
/etc/ansible/stage_hosts /etc/ansible/deployment.yml"
                   '''
                }
            }
        }
        stage('check stage website availability') {
            steps {
                 sh "sleep 200"
                 sh "curl -s -o /dev/null -w \"%{http_code}\"
https://stage.edenboutique.space"
                script {
                    def response = sh(script: "curl -s -o /dev/null -w \"%
{http_code}\" https://stage.edenboutique.space", returnStdout: true).trim()
                    if (response == "200") {
                        slackSend(color: 'good', message: "The stage
petclinic website is up and running with HTTP status code ${response}.",
tokenCredentialId: 'slack')
                    } else {
                        slackSend(color: 'danger', message: "The stage
petclinic wordpress website appears to be down with HTTP status code $
{response}.", tokenCredentialId: 'slack')
                    }
                }
            }
        }
        stage('Request for Approval') {
            steps {
                timeout(activity: true, time: 10) {
                    input message: 'Needs Approval ', submitter: 'admin'
                }
```

```
            }
        }
        stage('Deploy to prod') {
            steps {
                sshagent(['ansible-key']) {
                    sh '''
                         ssh -t -t -o StrictHostKeyChecking=no -o
ProxyCommand="ssh -W %h:%p -o StrictHostKeyChecking=no ec2-user@$
{BASTION_IP}" ec2-user@${ANSIBLE_IP} "ansible-playbook -i
/etc/ansible/prod_hosts /etc/ansible/deployment.yml"
                    '''
                }
            }
        }
        stage('check prod website availability') {
            steps {
                 sh "sleep 200"
                 sh "curl -s -o /dev/null -w \"%{http_code}\"
https://www.edenboutique.space"
                script {
                    def response = sh(script: "curl -s -o /dev/null -w \"%
{http_code}\" https://www.edenboutique.space", returnStdout: true).trim()
                    if (response == "200") {
                        slackSend(color: 'good', message: "The prod petclinic
website is up and running with HTTP status code ${response}.",
tokenCredentialId: 'slack')
                    } else {
                        slackSend(color: 'danger', message: "The prod
petclinic wordpress website appears to be down with HTTP status code $
{response}.", tokenCredentialId: 'slack')
                    }
                }
            }
        }
    }
}
```

Dockerfile explanation

This docker file is executed when the docker build stage of the pipeline is triggered.
here it sets the base image, locates the .war file and copies it to the tomcat webapp directory, sets the directory it would run the application, install dependencies and run the application.

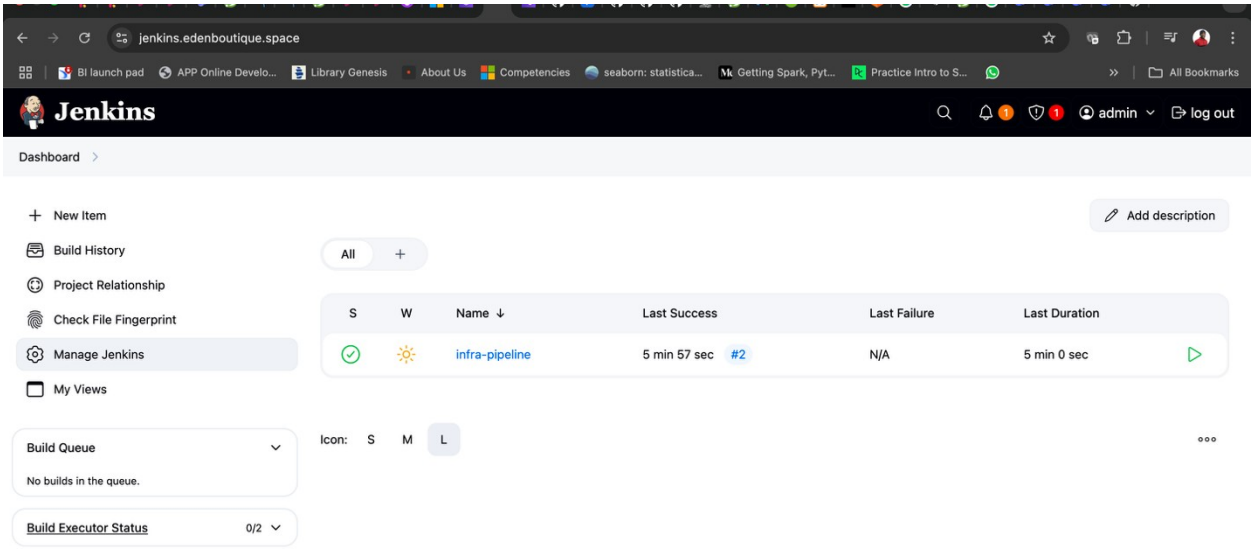
```
FROM openjdk:8-jre-slim
FROM ubuntu
FROM tomcat
#copy war file on the container
COPY **/*.war /usr/local/tomcat/webapps
WORKDIR  /usr/local/tomcat/webapps
RUN apt update -y && apt install curl -y
RUN curl -O https://download.newrelic.com/newrelic/java-agent/newrelic-
agent/current/newrelic-java.zip && \
    apt-get install unzip -y  && \
    unzip newrelic-java.zip -d  /usr/local/tomcat/webapps
WORKDIR /usr/local/tomcat/webapps
COPY newrelic.yml /usr/local/tomcat/webapps/newrelic/newrelic.yml
```

ENTRYPOINT ["java", "-javaagent:/usr/local/tomcat/webapps/newrelic/newrelic.jar", "-jar", "spring-petclinic-2.4.2.war", "--server.port=8080

J. JENKINS PIPELINE SETUP FOR APPLICATION AND APPLICATION DEPLOYMENT

To setup the Jenkins pipeline, we need to install plugins,setup nexus,sonarqube and all other dependencies.

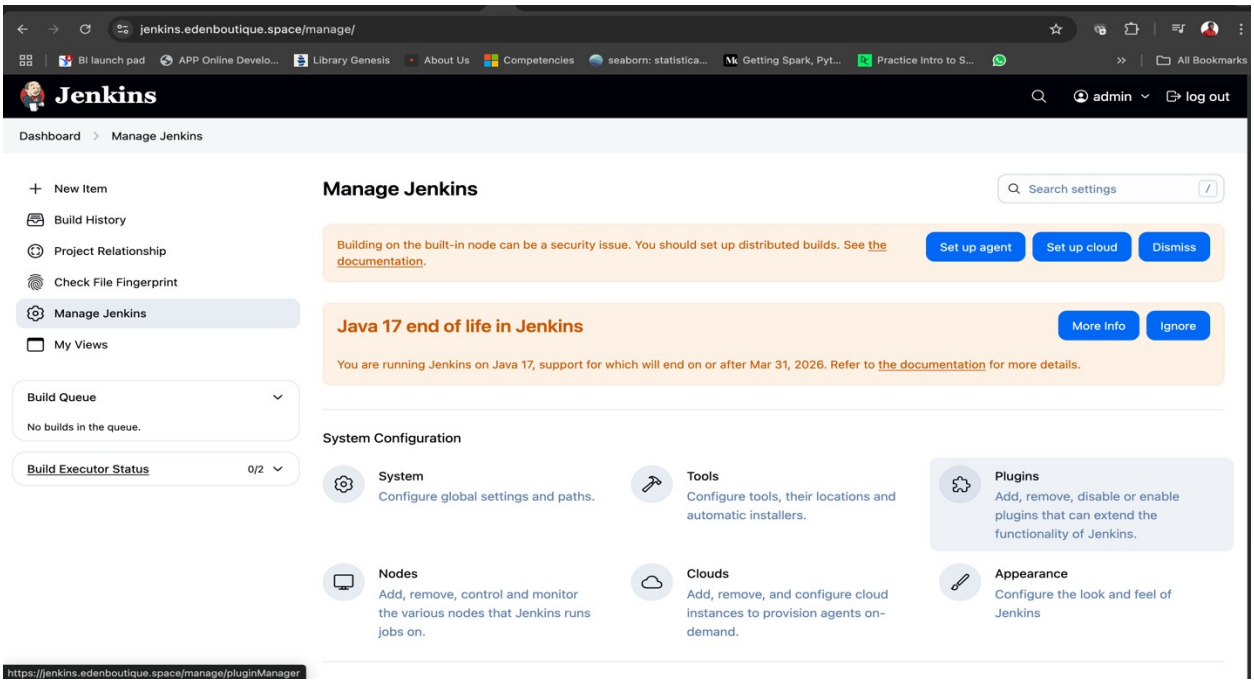
First go to manage Jenkins on your Jenkins url



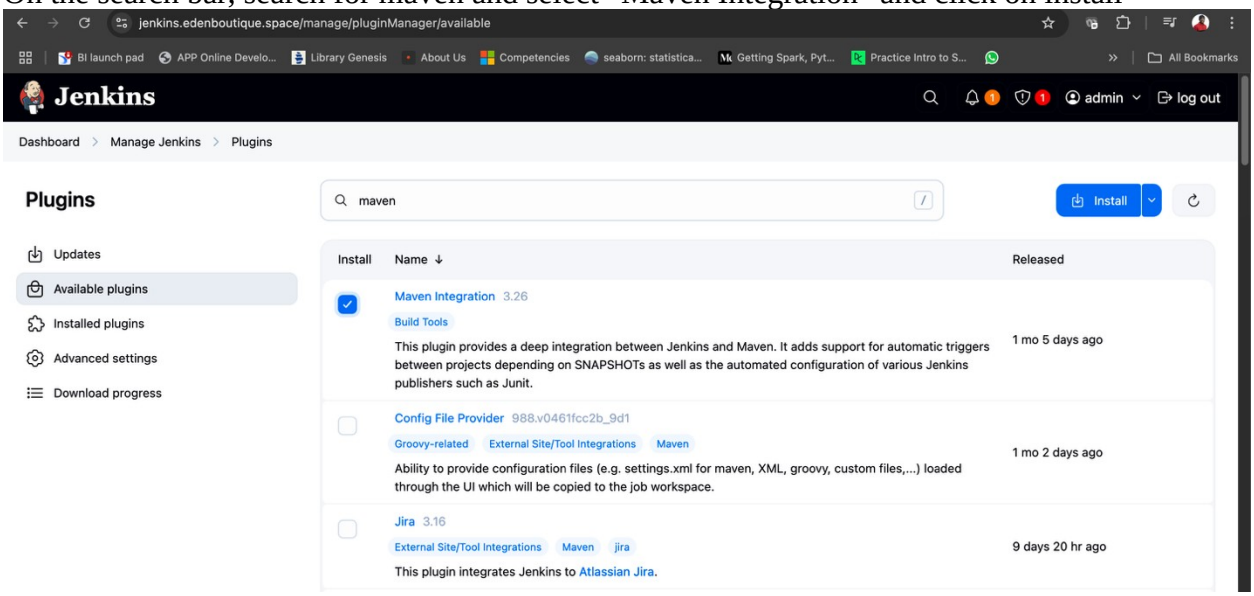
see resources deployed by Jenkins pipeline

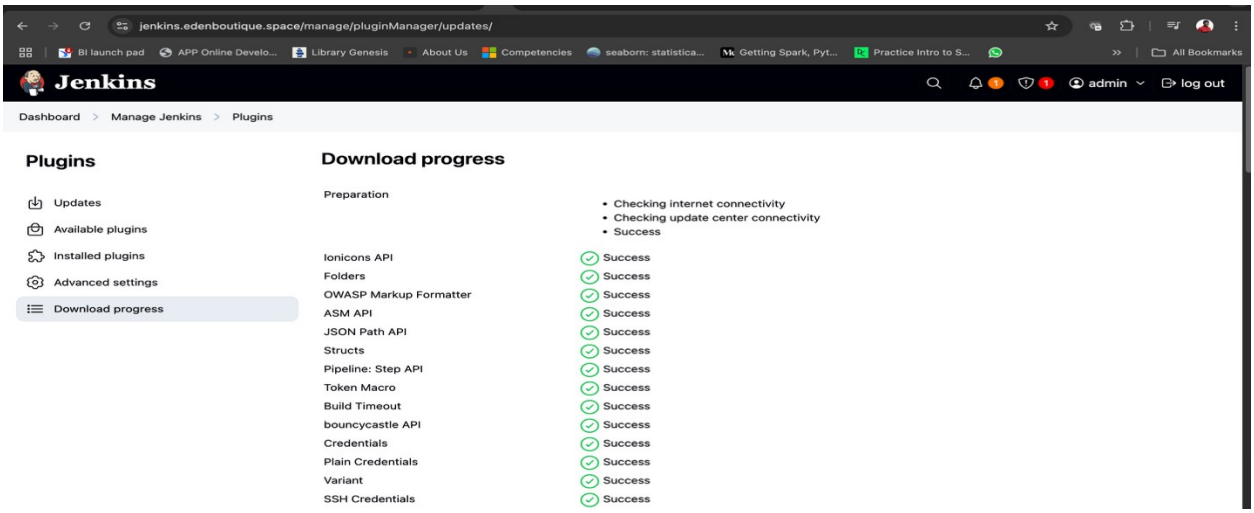
INSTALL MAVEN PLUGIN

Next, click on plugins, here we want to install maven



On the search bar, search for maven and select “Maven Integration” and click on install

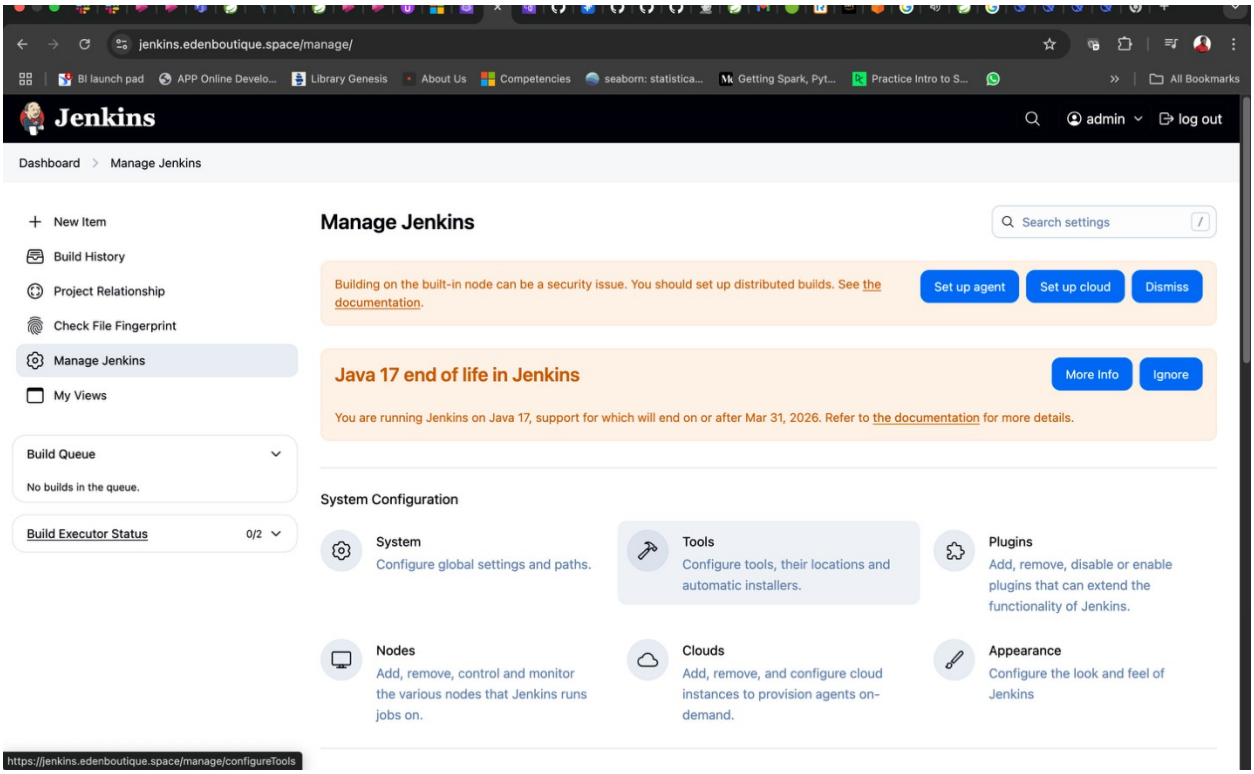




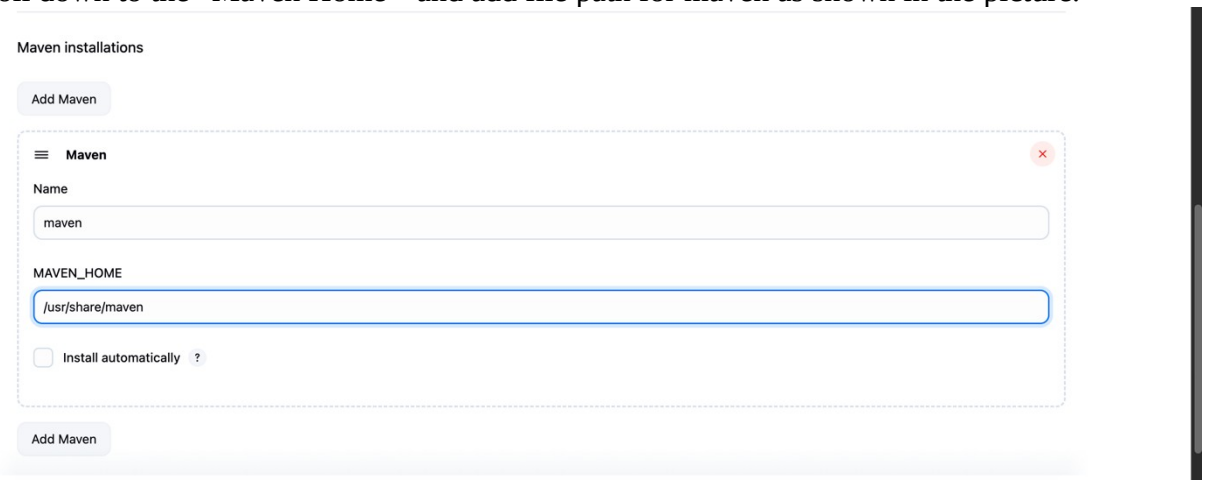
ADD FILE PATH FOR MAVEN AND JAVA

Next we need to add a file path for maven and java to Jenkins tool so as to use when building our pipeline. To get the file path where maven and java is intalled, ssh into your Jenkins server and type the command "mvn --version" you can see that item 2 is the maven file path awhile item 3 points to the java path. Copy them and add to the tool section of the Jenkins

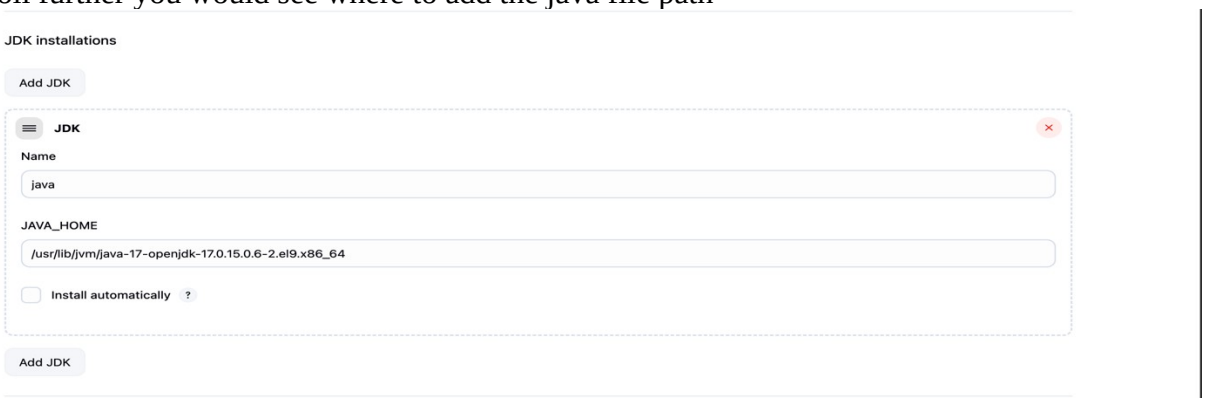
Go to the tool section



Scroll down to the “Maven Home ” and add file path for maven as shown in the picture.



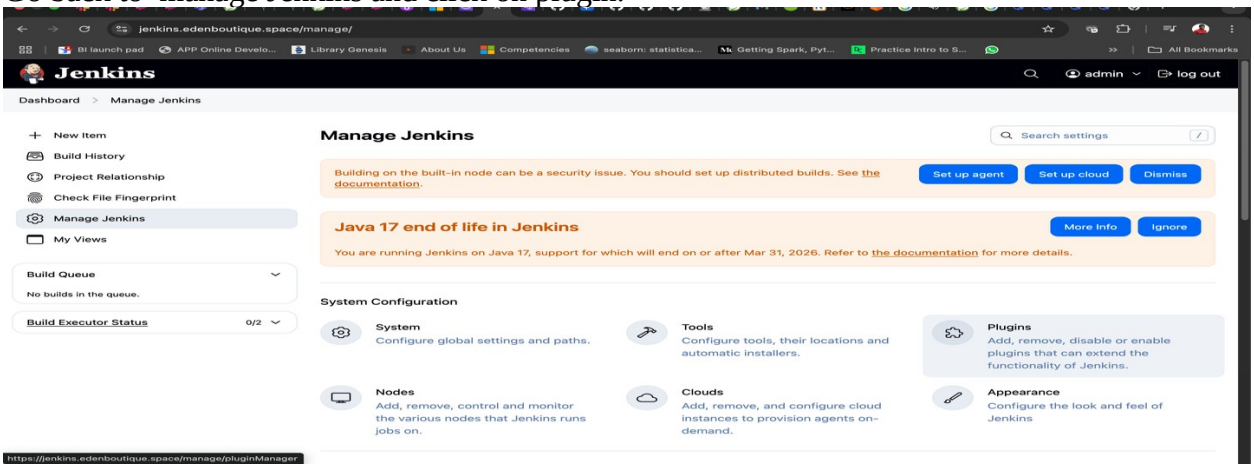
Scroll further you would see where to add the java file path



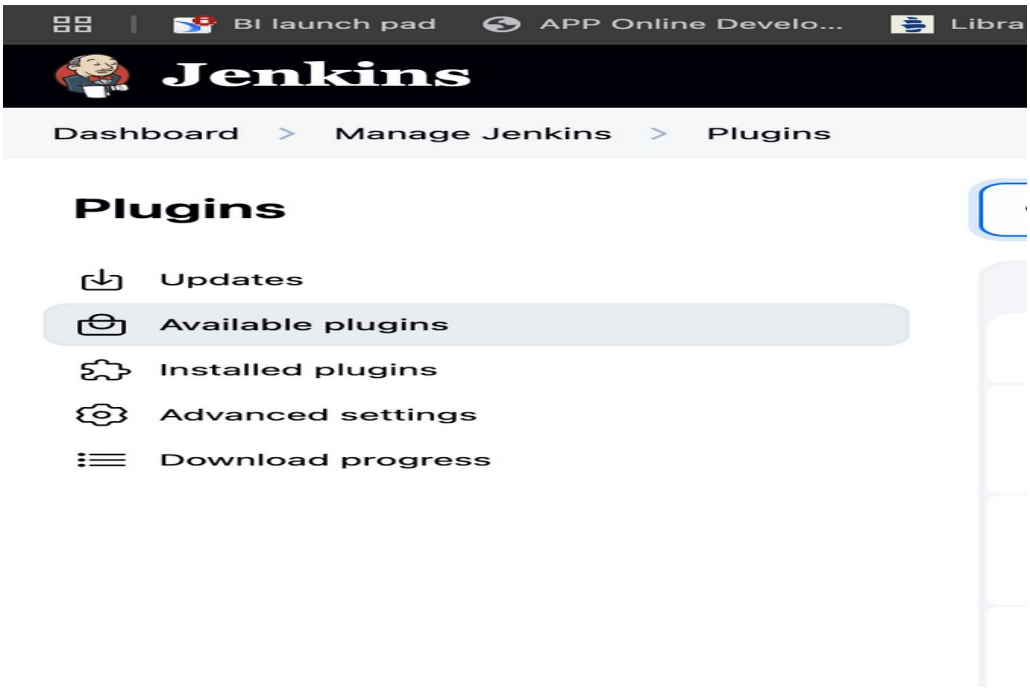
INSTALL SSH AGENT

Here we need to install the SSHagent so as Jenkins can have access to ssh into other servers in the pipeline.

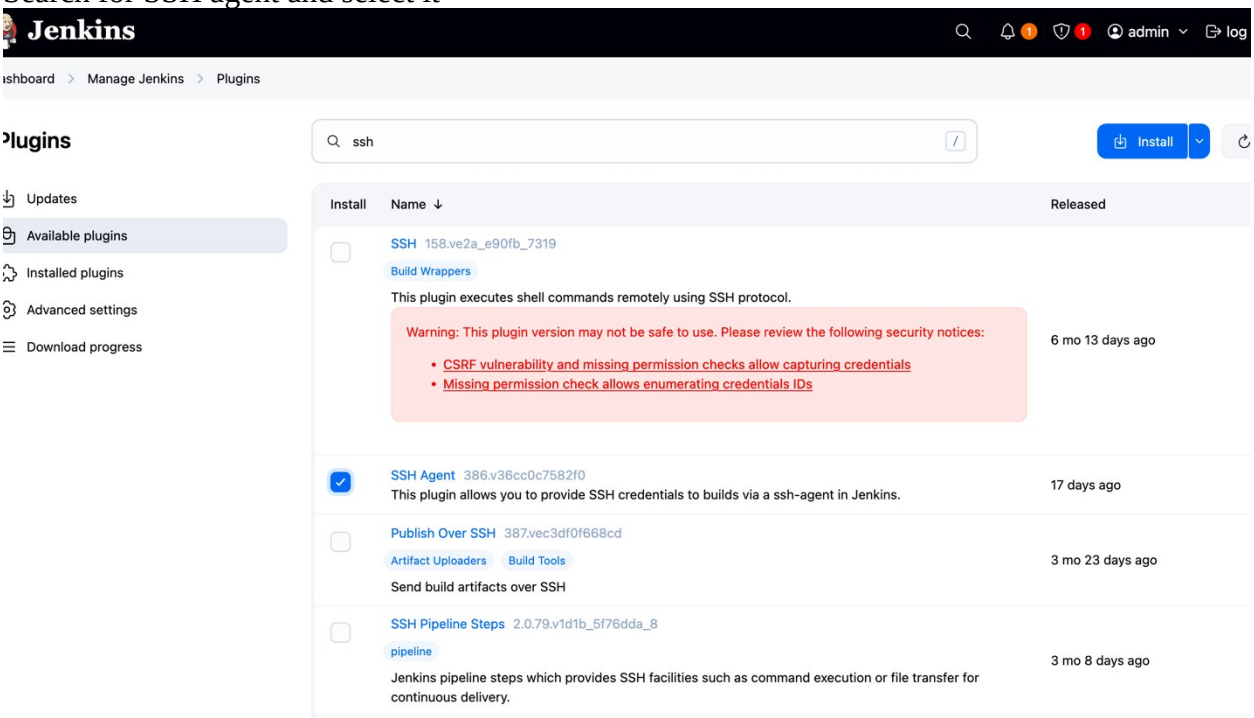
Go back to manage Jenkins and click on plugin.



Go to available plugin



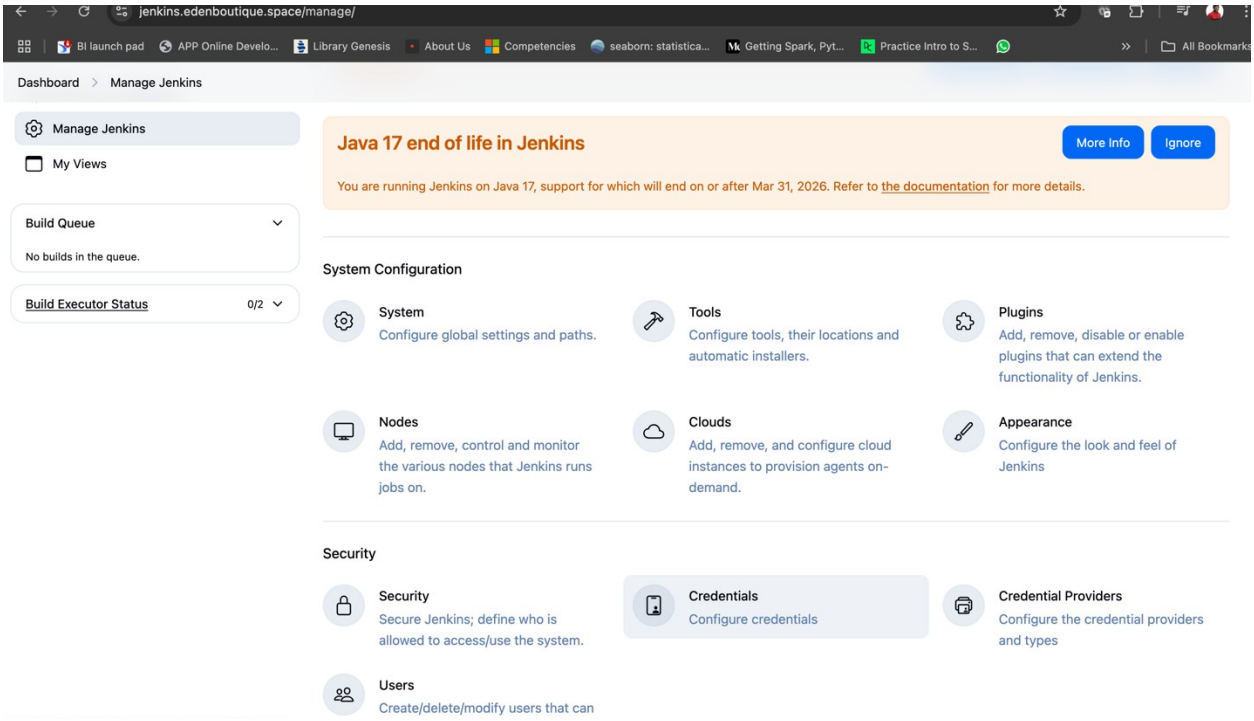
Search for SSH agent and select it



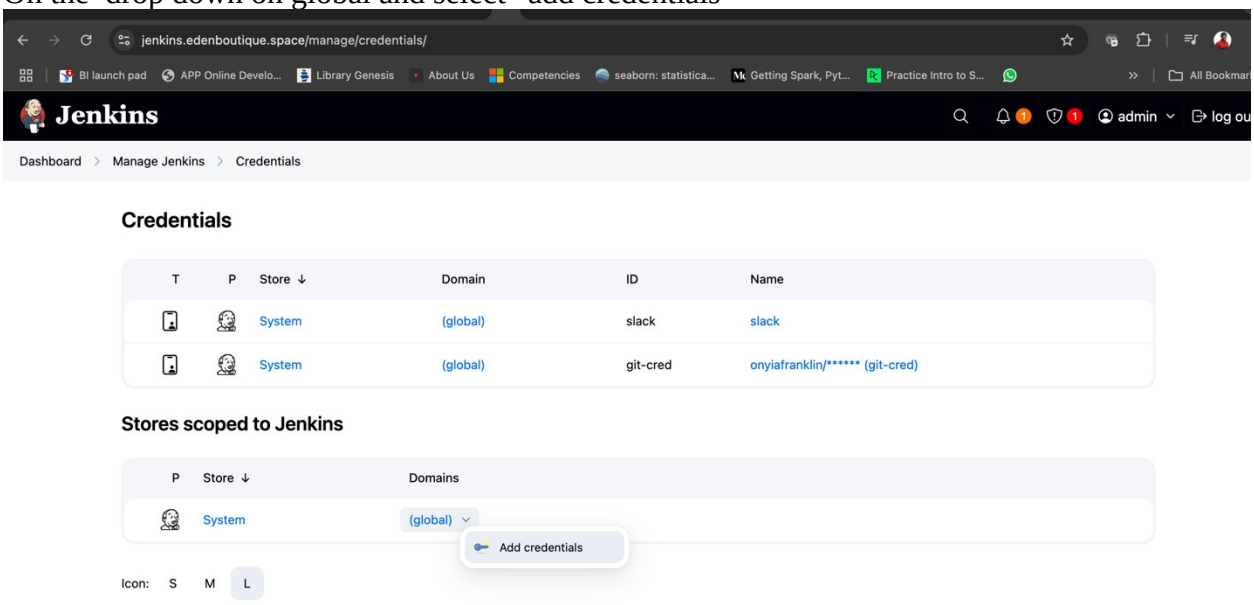
ADD THE ANSIBLE KEY CREDENTIALS

Here we need to add the username with private key that Jenkins would use to ssh into other servers. To do that,

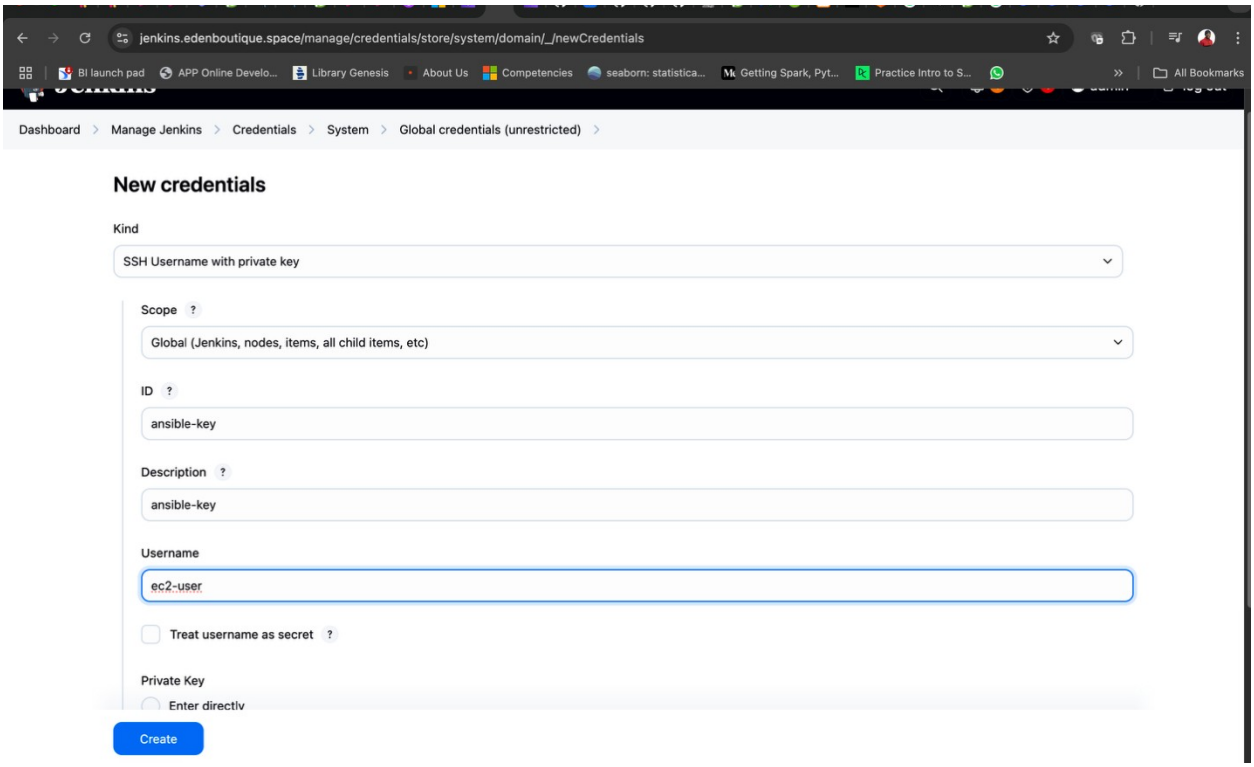
Go to credentials



On the drop down on global and select “add credentials”



On the kind, select “ssh username with private key”
add EC2-user as the user then go and locate the private key



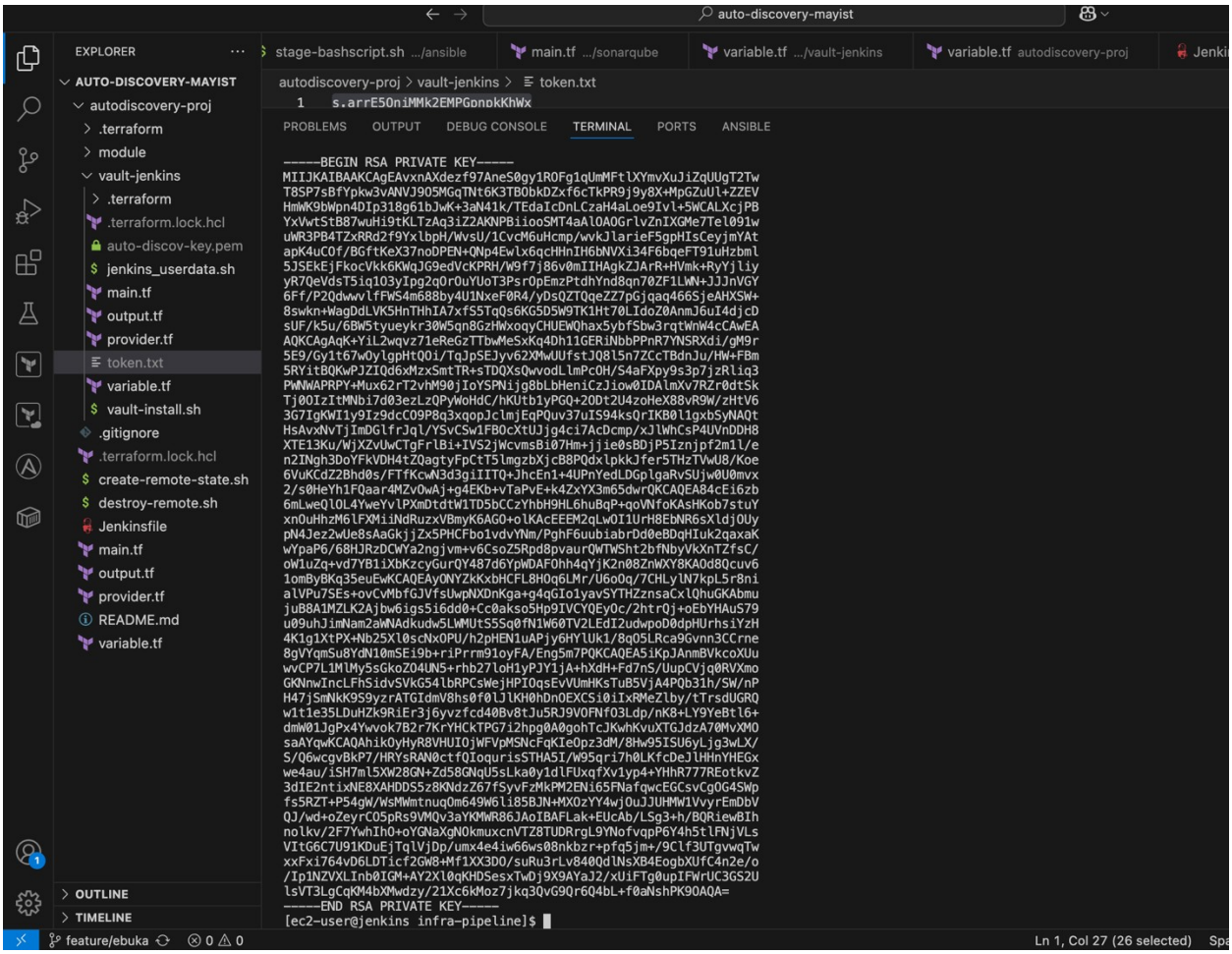
To locate the private key, ssh into your Jenkins server , cd into the Jenkins workspace cd /var/lib/Jenkins/workspace/(your pipeline name infra-pipeline) and list all file there you would see your pem key“auto-discov-key.pem”

```
OS name: "linux", version: "5.14.0-427.68.1.el9_4.x86_64", arch: "amd64", family: "unix"
[ec2-user@jenkins ~]$ cd /var/lib/jenkins/workspace/infra-pipeline
[ec2-user@jenkins infra-pipeline]$ ls -al
total 56
drwxr-xr-x.  7 root jenkins 4096 May 31 10:18 .
drwxr-xr-x.  4 root jenkins   54 May 31 10:14 ..
-r--r-----.  1 root jenkins 3243 May 31 10:18 auto-discov-key.pem
drwxr-xr-x.  2 root jenkins   29 May 31 10:18 checkov-results.txt
-rw-r--r--.  1 root jenkins 1533 May 31 10:14 create-remote-state.sh
-rw-r--r--.  1 root jenkins 2376 May 31 10:14 destroy-remote.sh
drwxr-xr-x.  8 root jenkins  162 May 31 10:17 .git
```

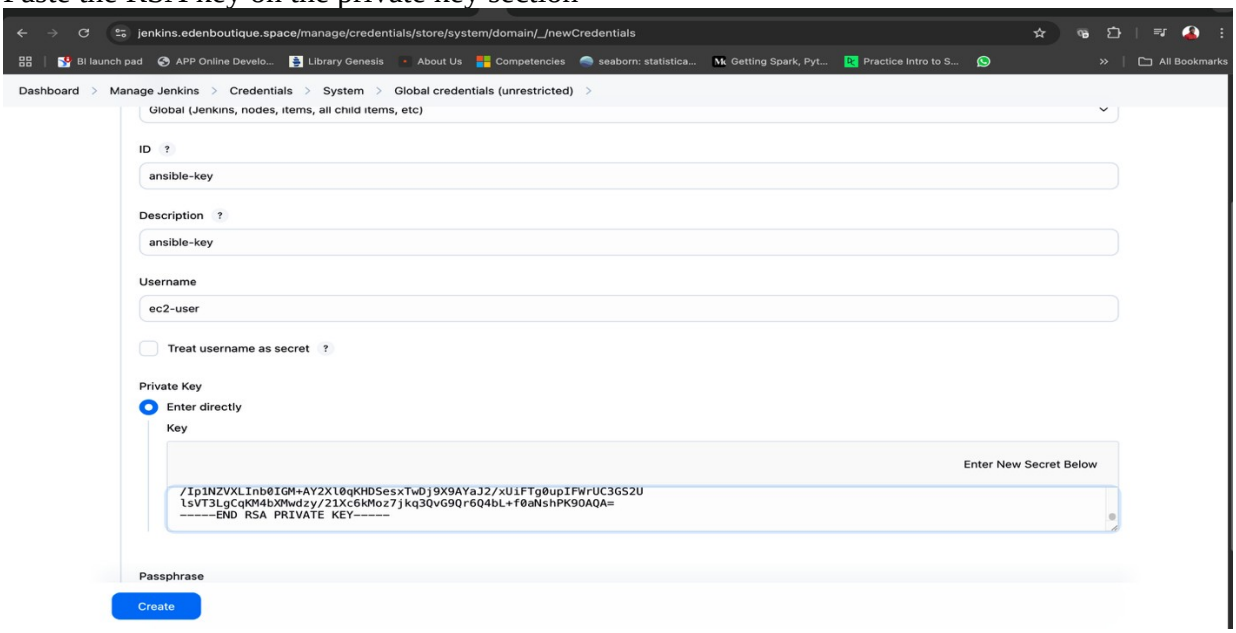
cat the key so that you can copy the key

```
-rw-r--r--.  1 root jenkins 6604 May 31 10:18 .terraform.lock.hcl
-rw-r--r--.  1 root jenkins  300 May 31 10:18 variable.tf
drwxr-xr-x.  2 root jenkins  127 May 31 10:14 vault-jenkins
[ec2-user@jenkins infra-pipeline]$ sudo cat auto-discov-key.pem
0                                                    Ln 1, Col 27 (26 selected)  Spaces:
```

Copy the RSA private key



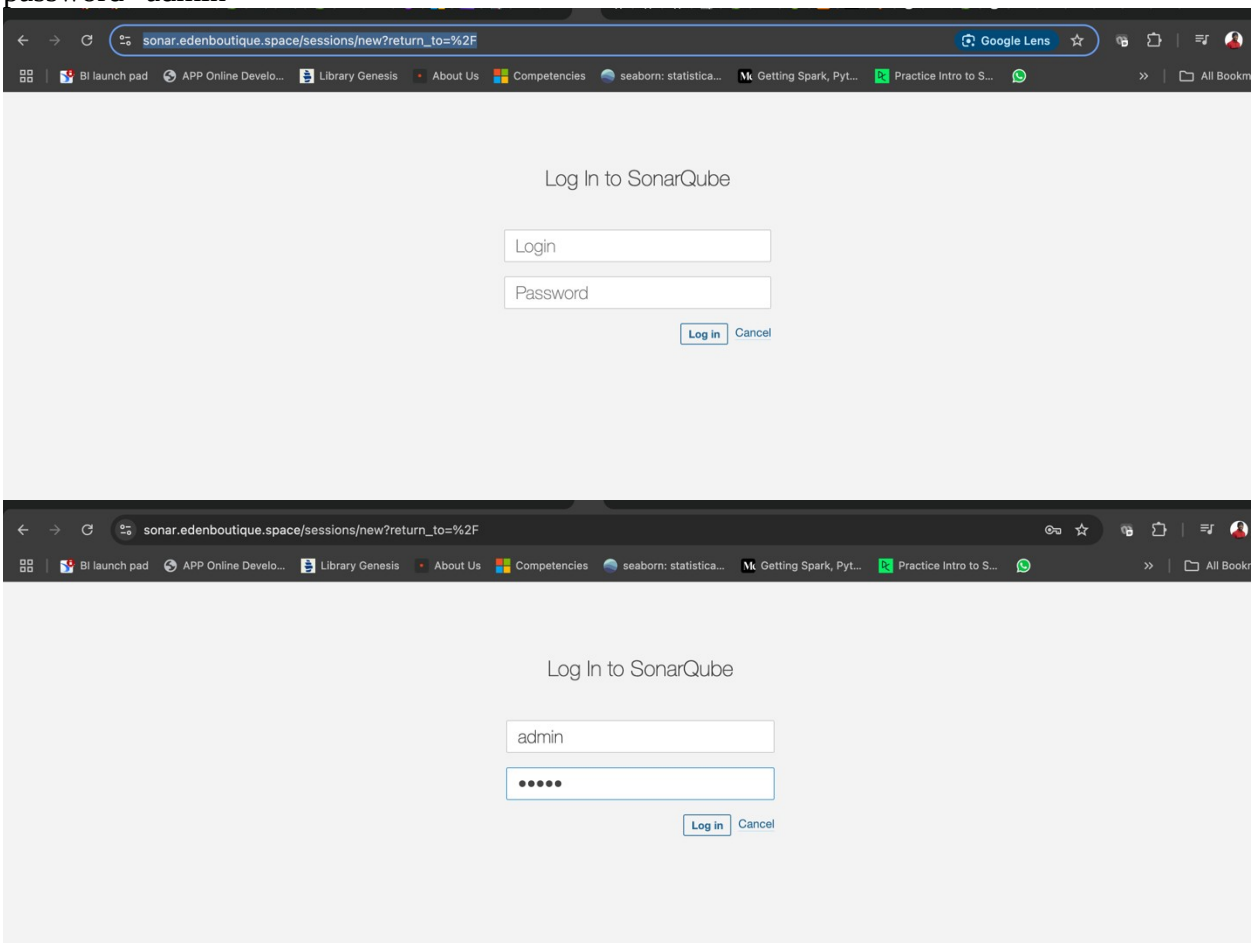
Paste the RSA key on the private key section



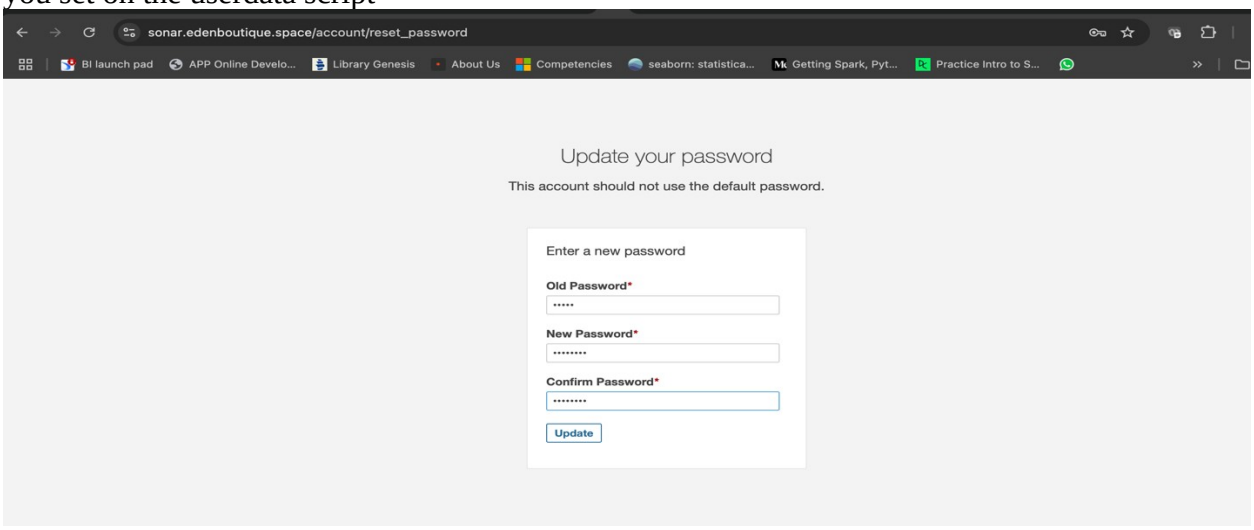
CONFIGURE SONARQUBE

To configure sonarqube, we need to create a webhook so that sonarqube can connect to

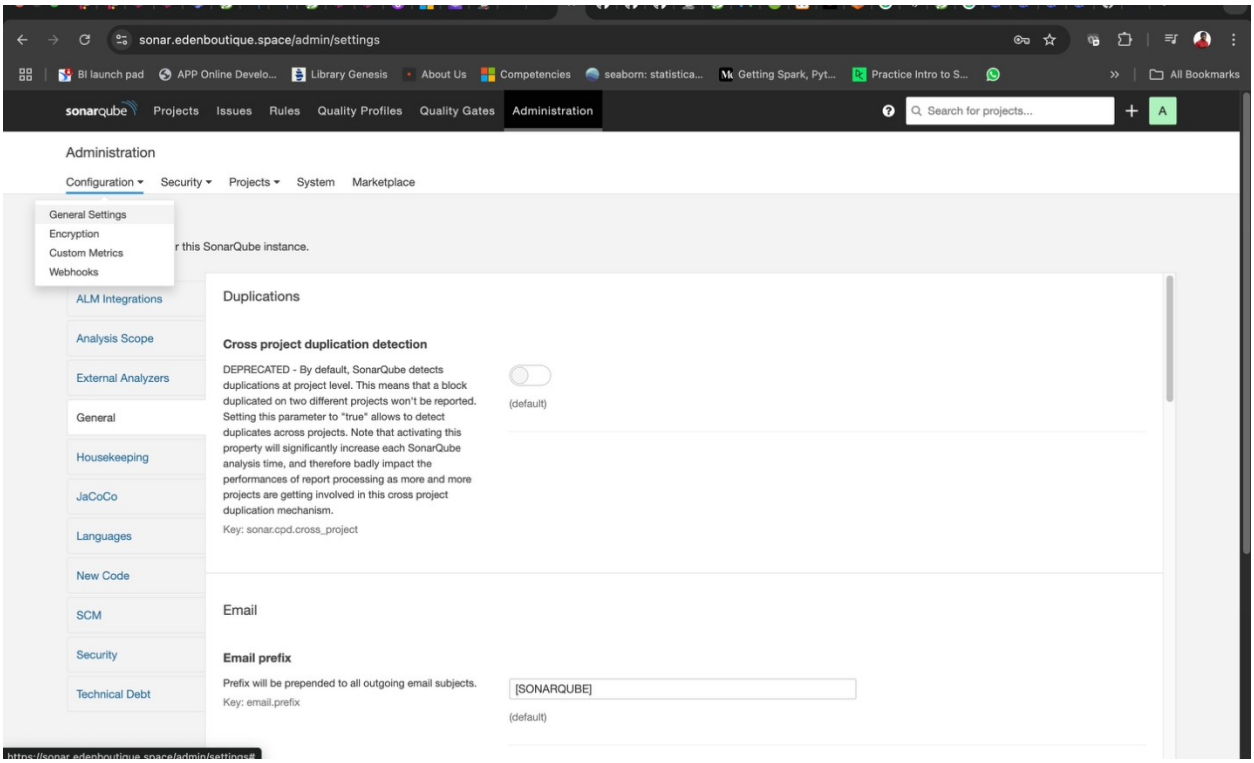
Jenkins as well as create a webhook on Jenkins so that Jenkins would be aware of the sonarqube aswell
First visit the sonarqube url and login with default credentials username“admin” password ”admin”



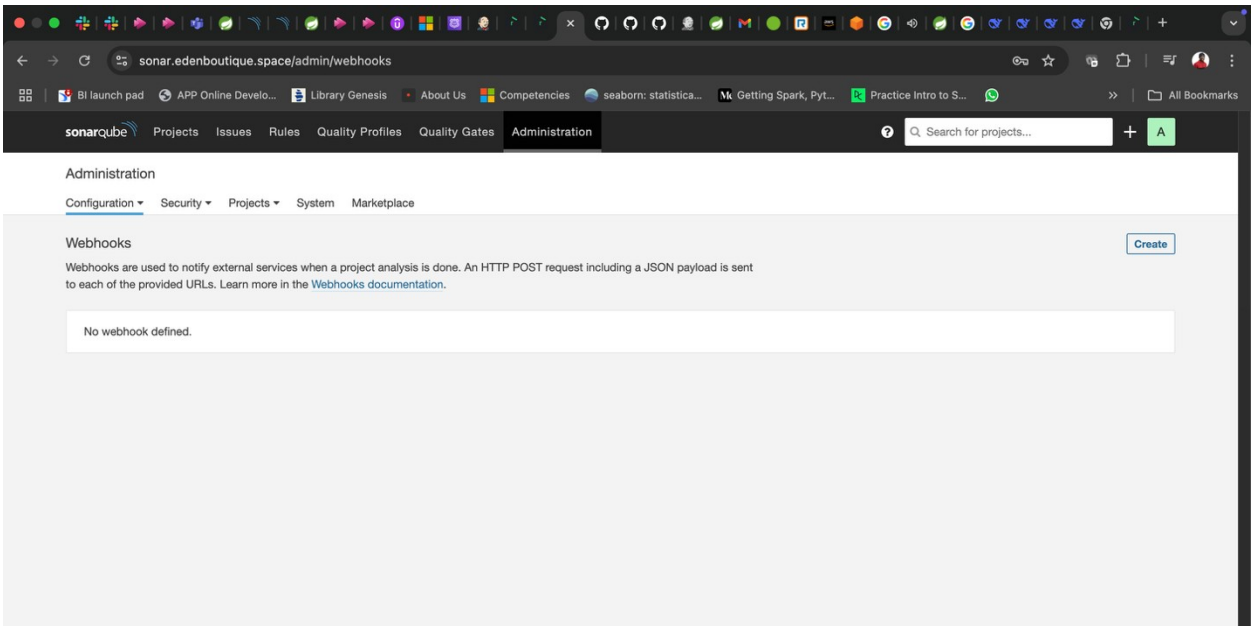
You would be prompted to change the password, here you can change it to the credentials you set on the userdata script



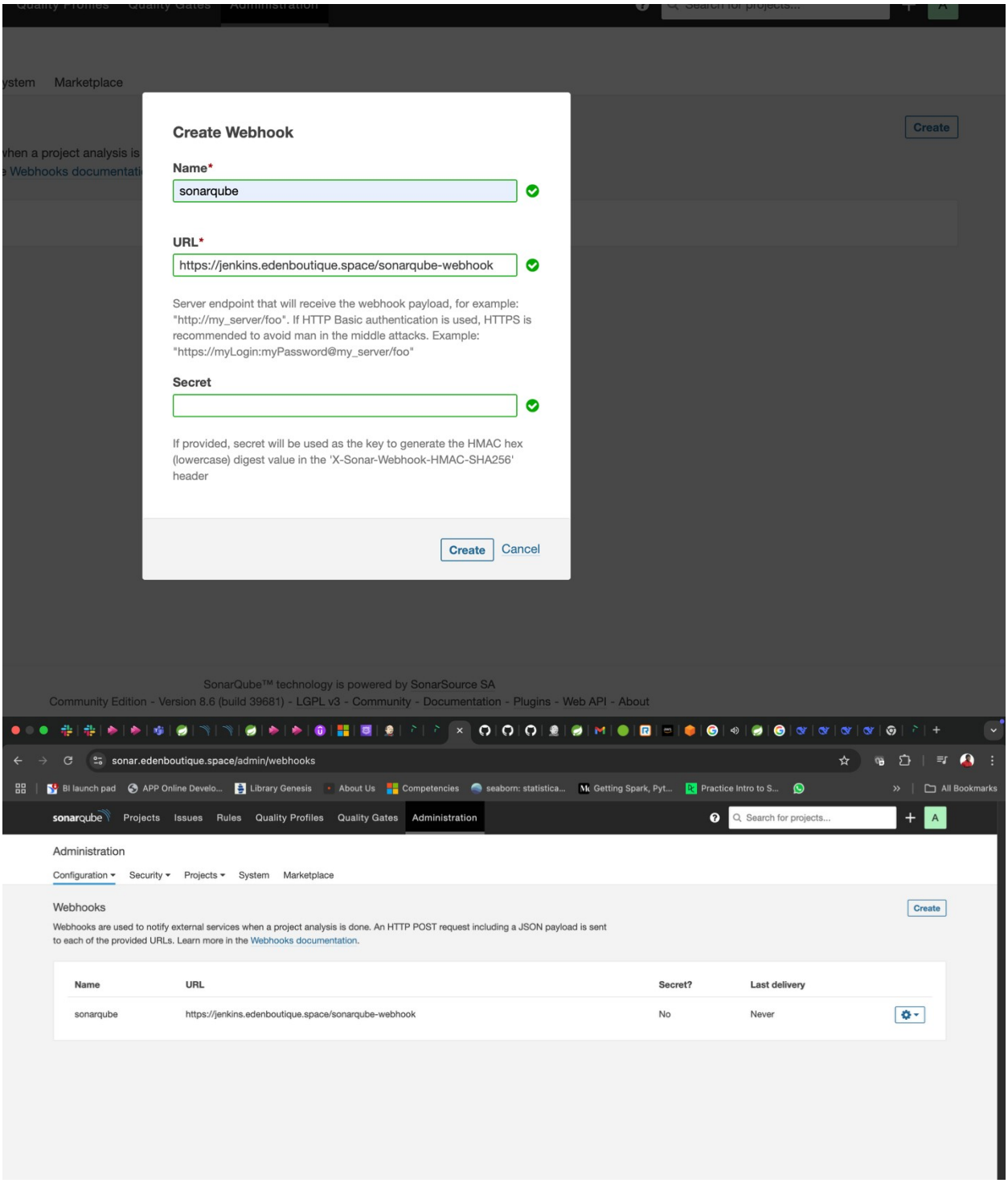
Go to configuration and click on “webhooks” then create a webhook



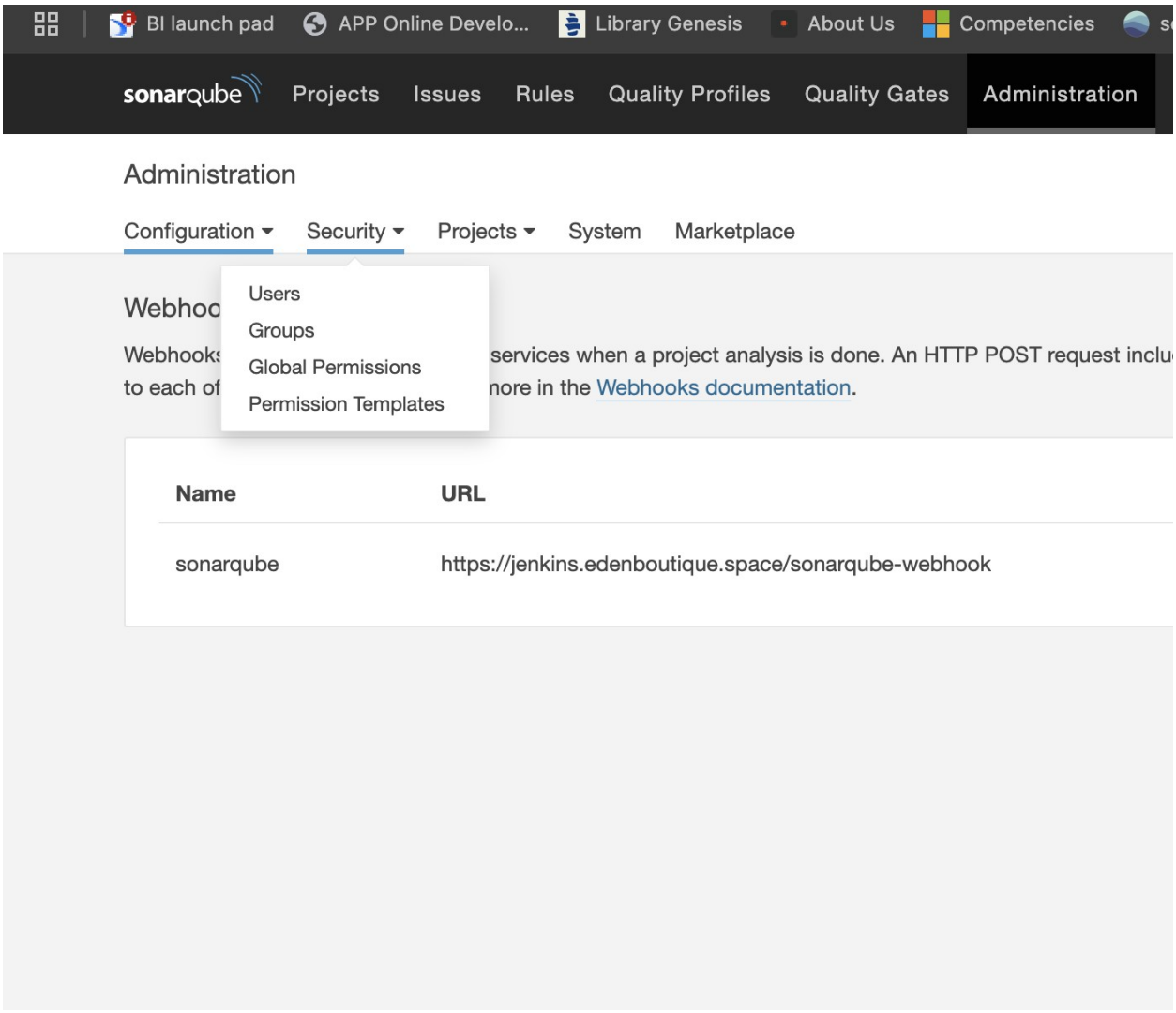
Click on create



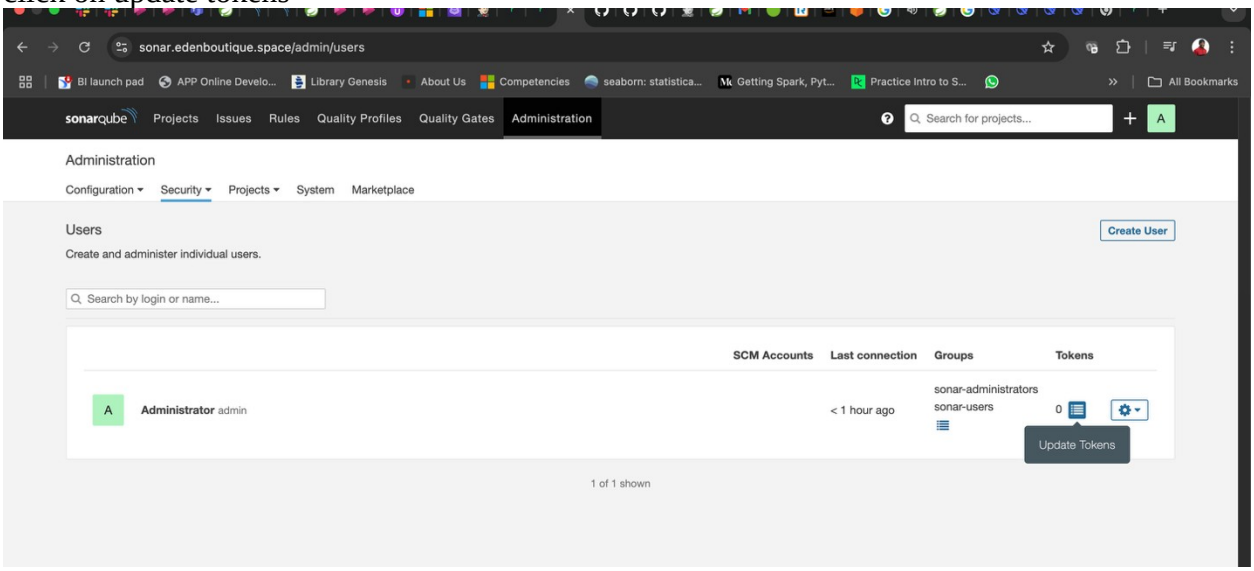
Fill in the "Create webhook " with Name and the url of the Jenkins as shown below.
Click on create



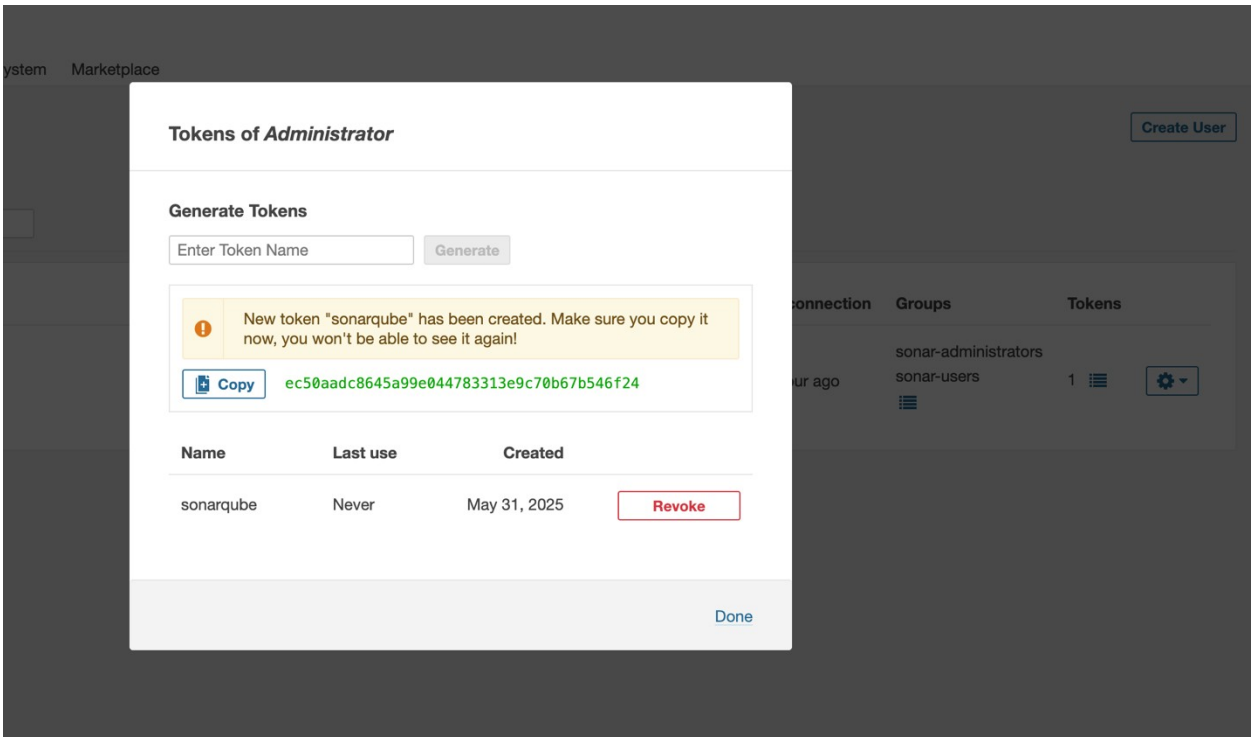
Next click on security and then users



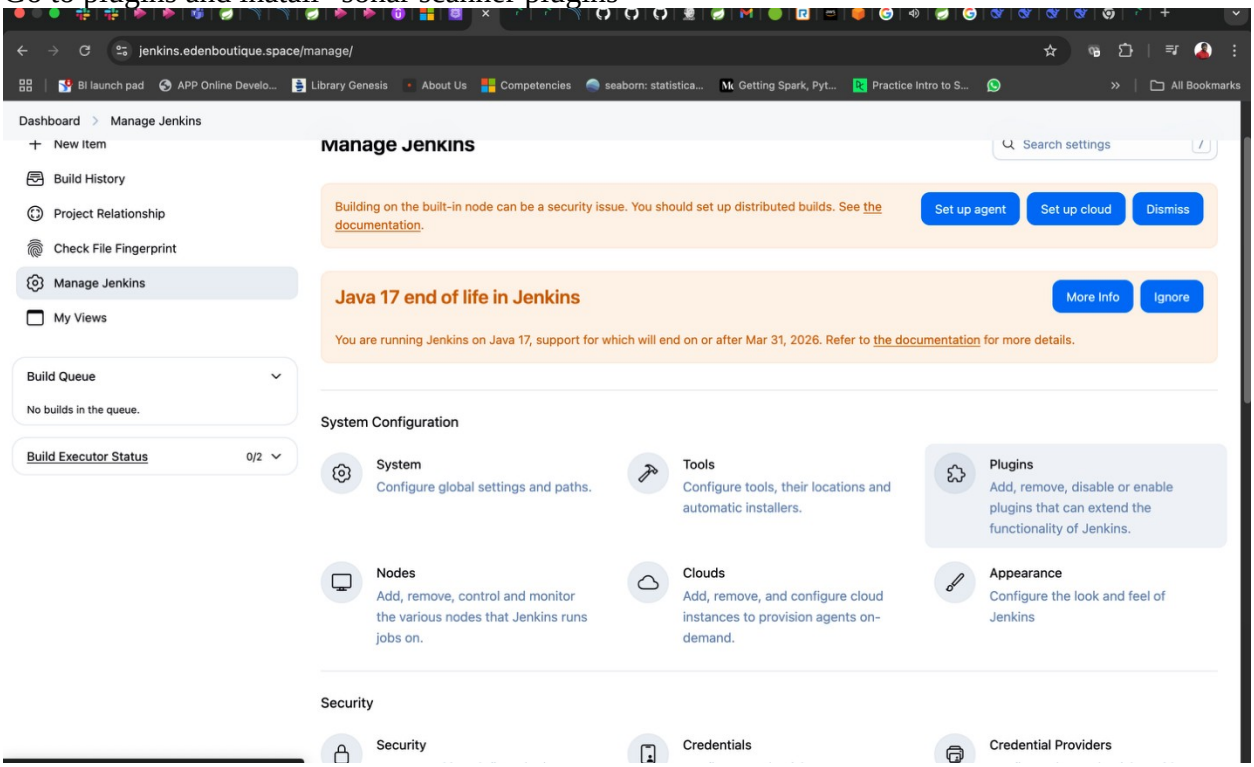
click on update tokens

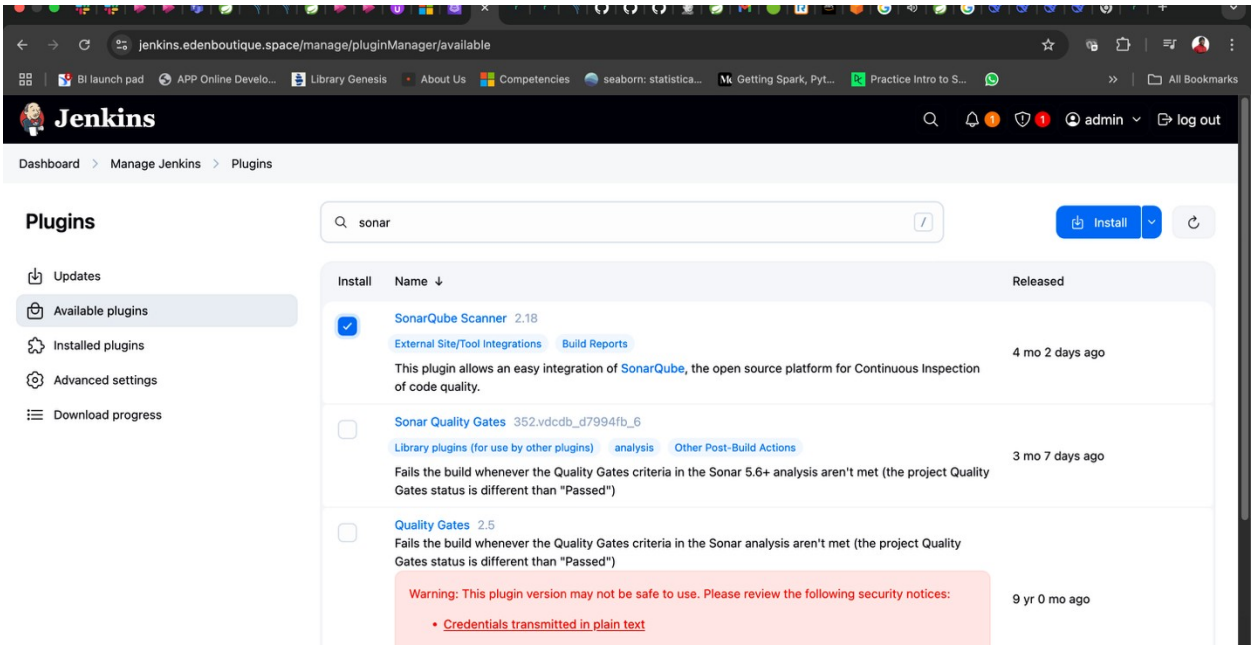


Enter token name an generate token, copy the created token

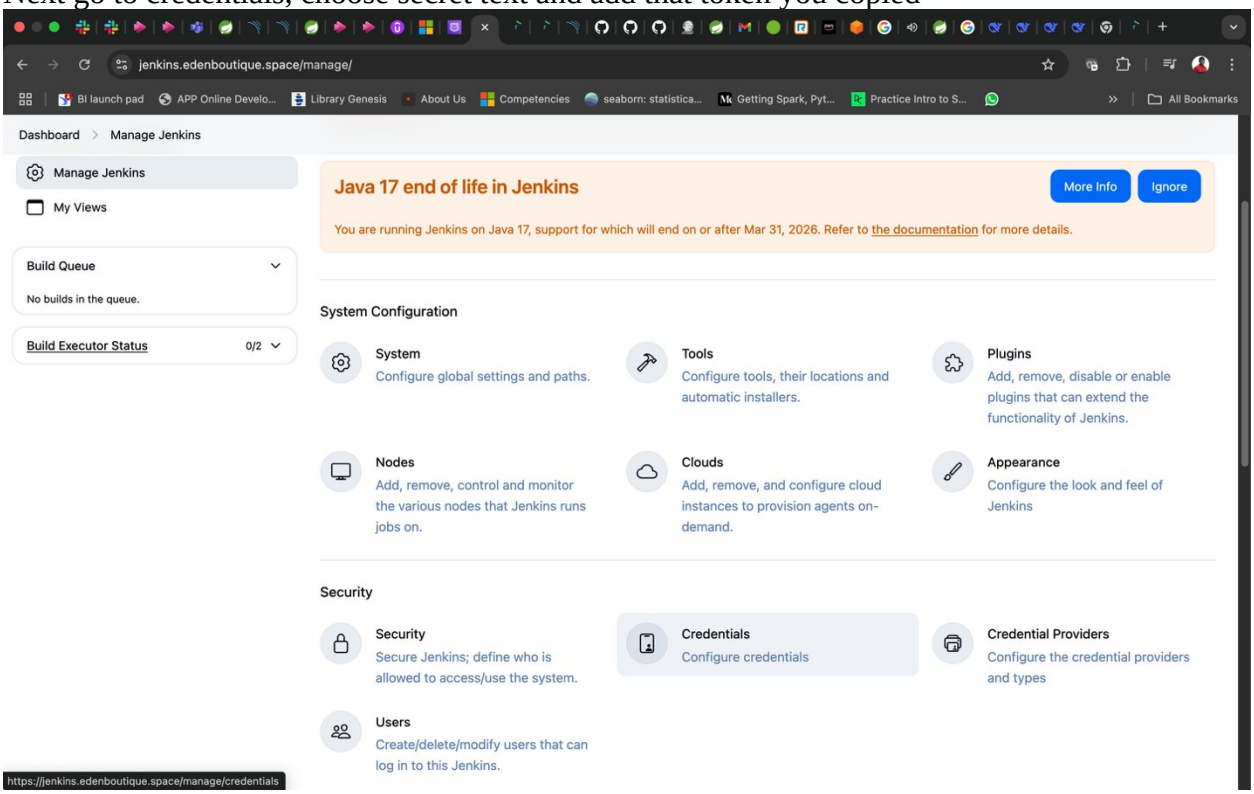


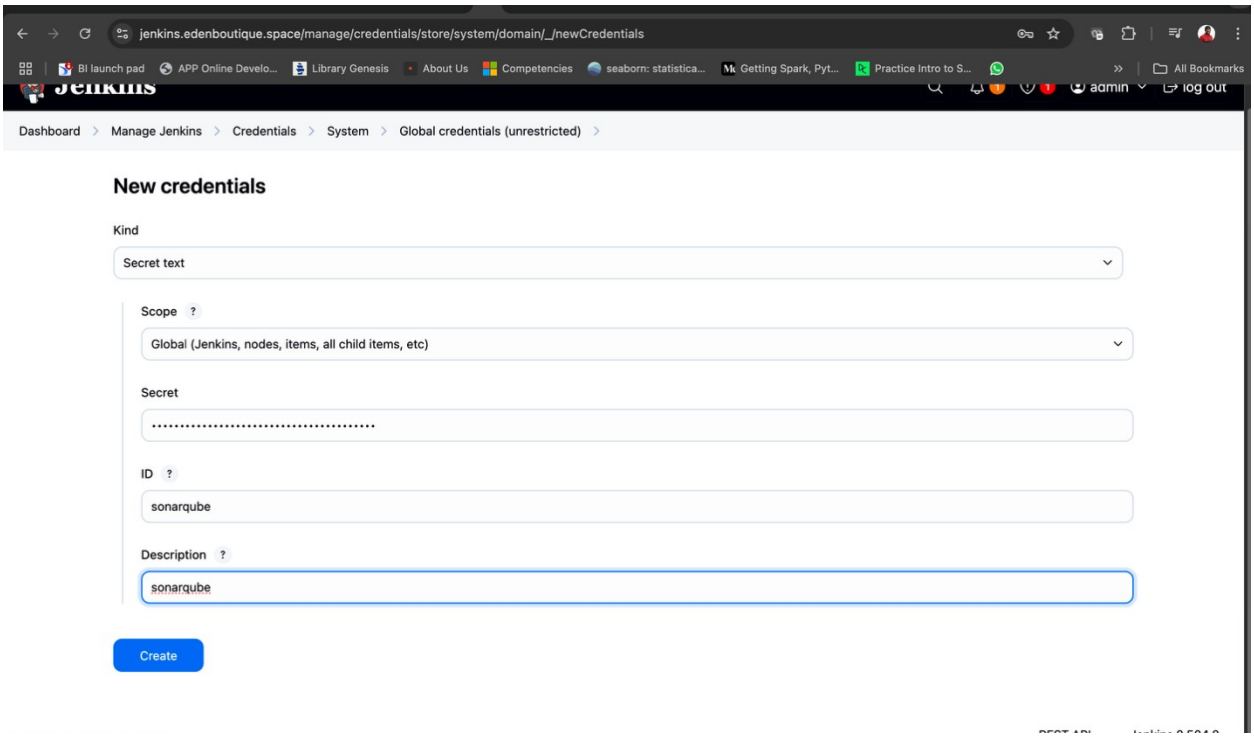
Go to plugins and inatall “sonar scanner plugins”



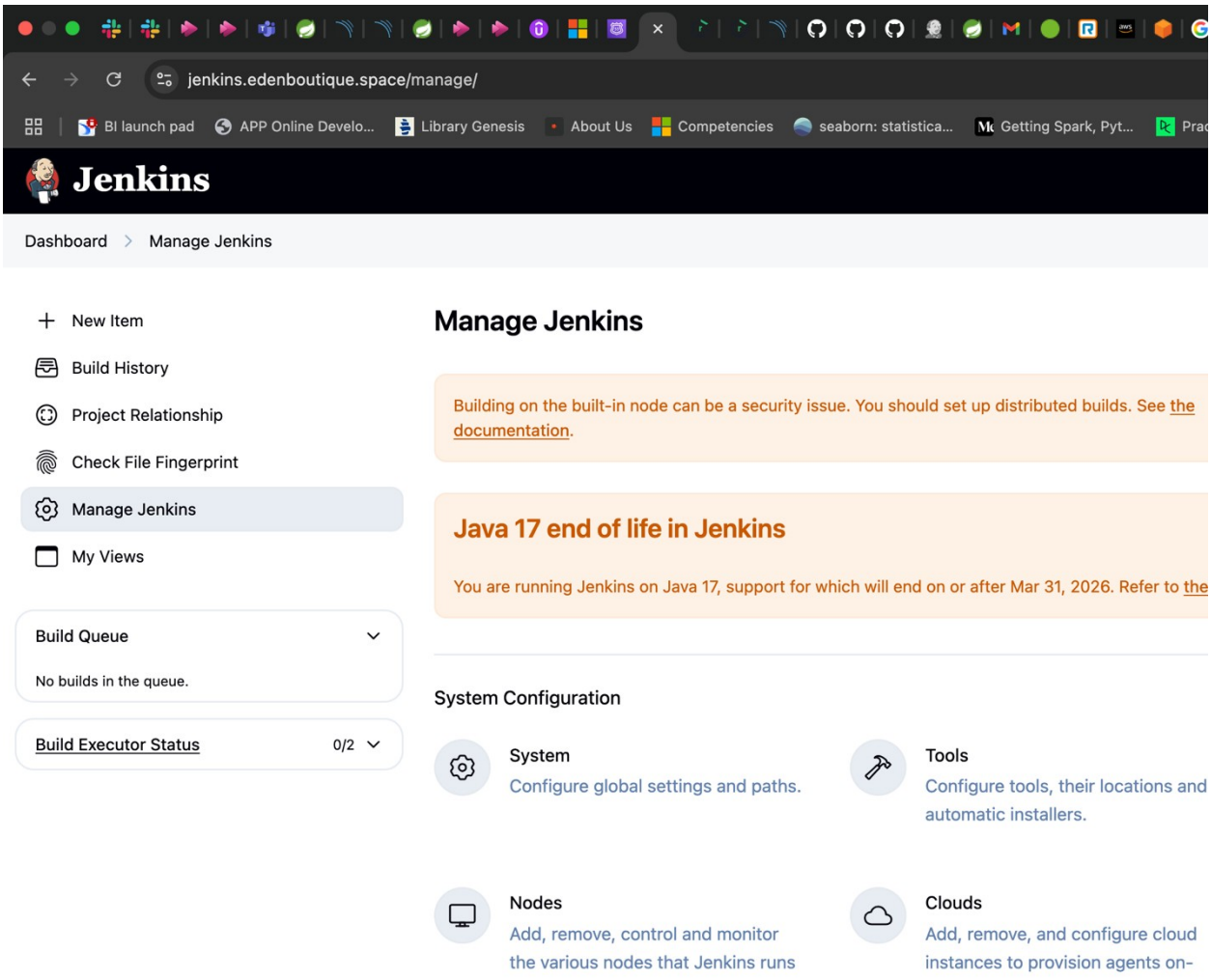


Next go to credentials, choose secret text and add that token you copied

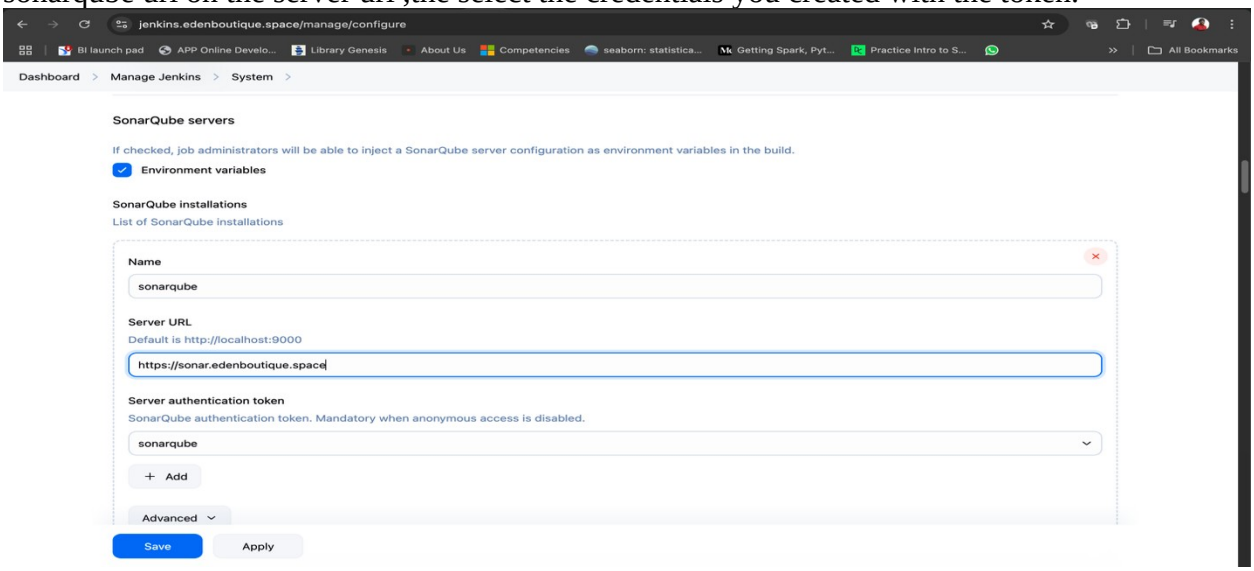




Go back to manage Jenkins and click on System



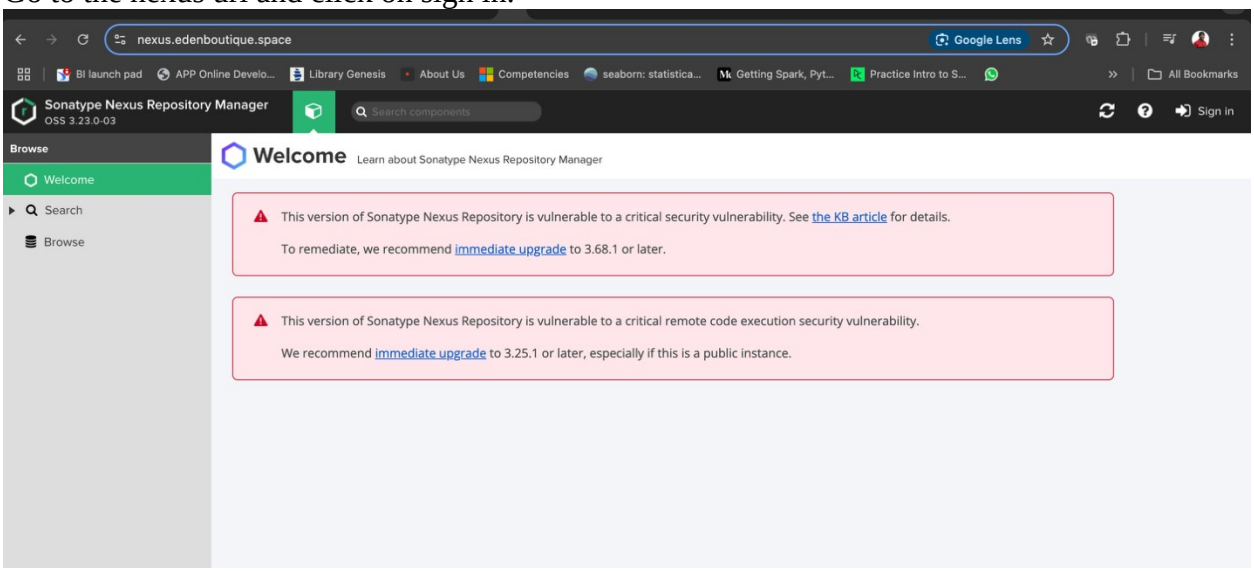
Scroll down to sonarqube servers, enable the environment variable and then add the sonarqube url on the server url ,the select the credentials you created with the token.



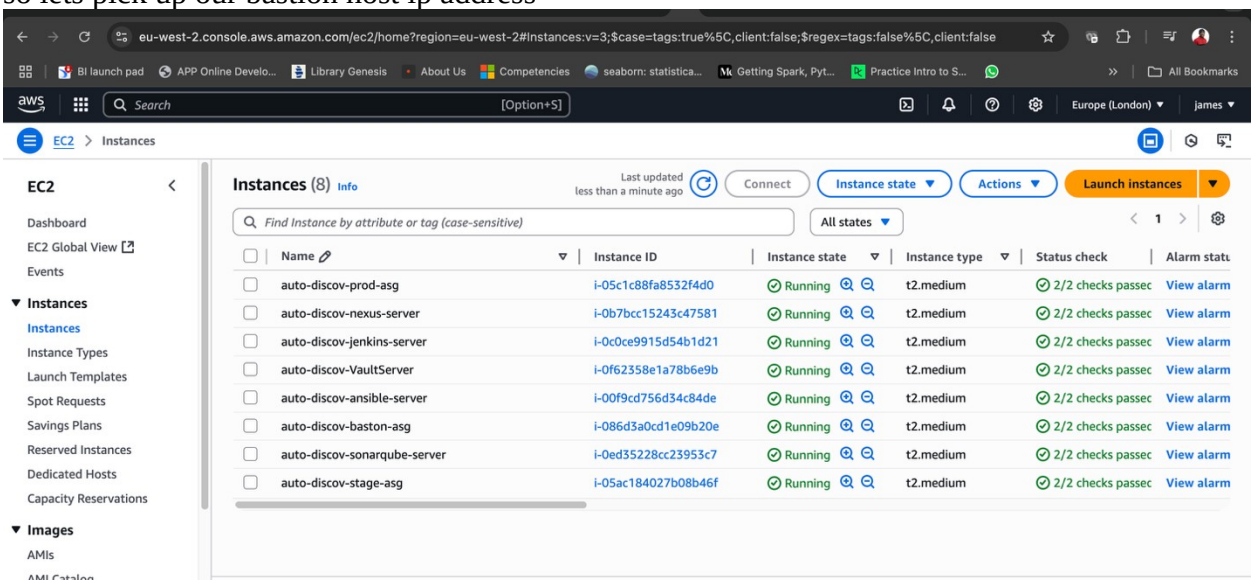
SET UP NEXUS SERVER

To setup our nexus server, we need to visit the url and creat two registries one which is

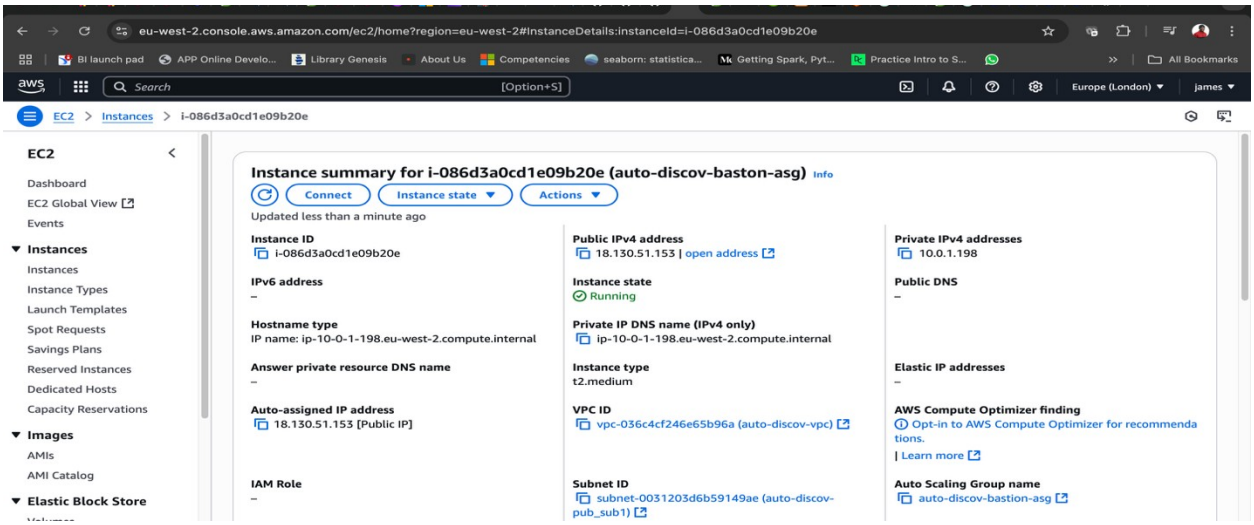
maven hosted to store the artefact generated from maven and the other which is docker hosted to store our docker image.
Go to the nexus url and click on sign in.



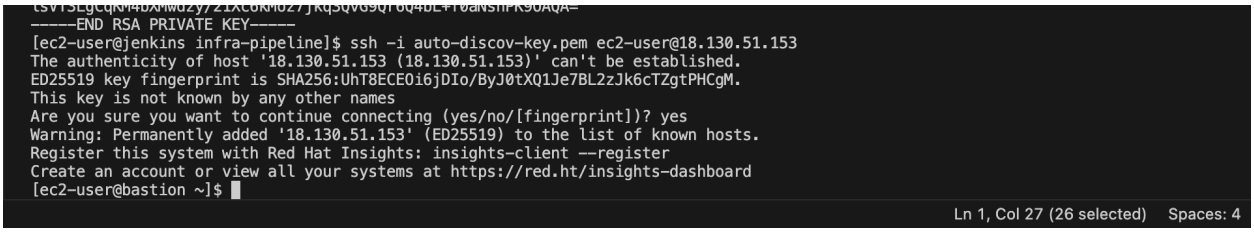
Remember, we setup nexus server security group in a way that we allow ssh from the bastion.
so lets pick up our bastion host ip address



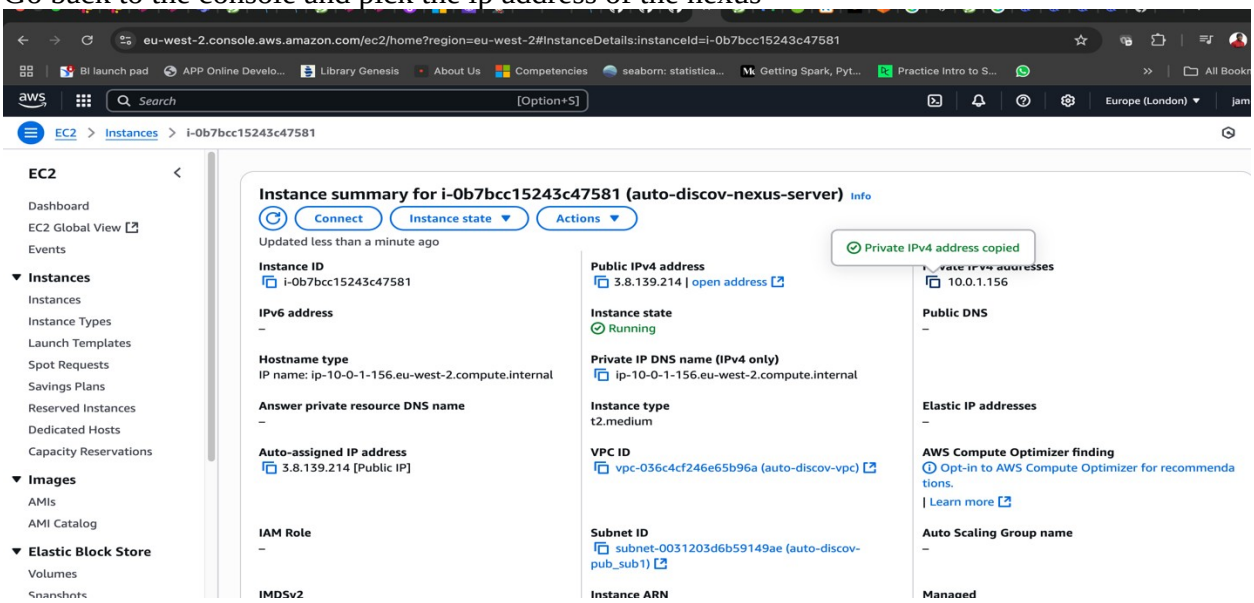
copy the ip address



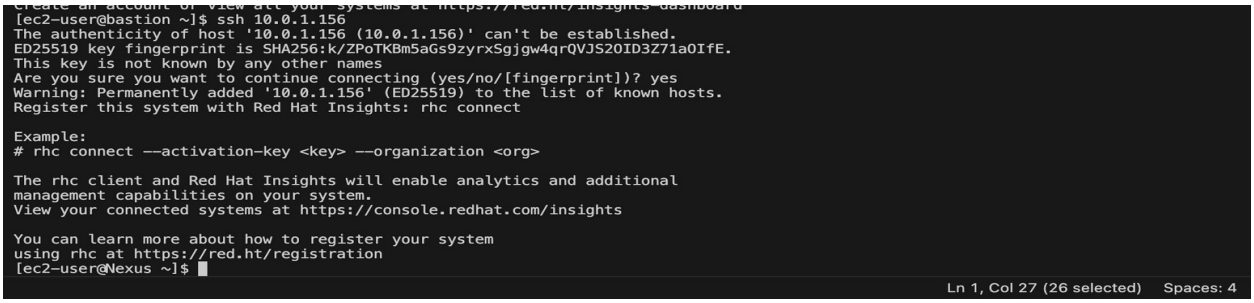
ssh into the bastion host



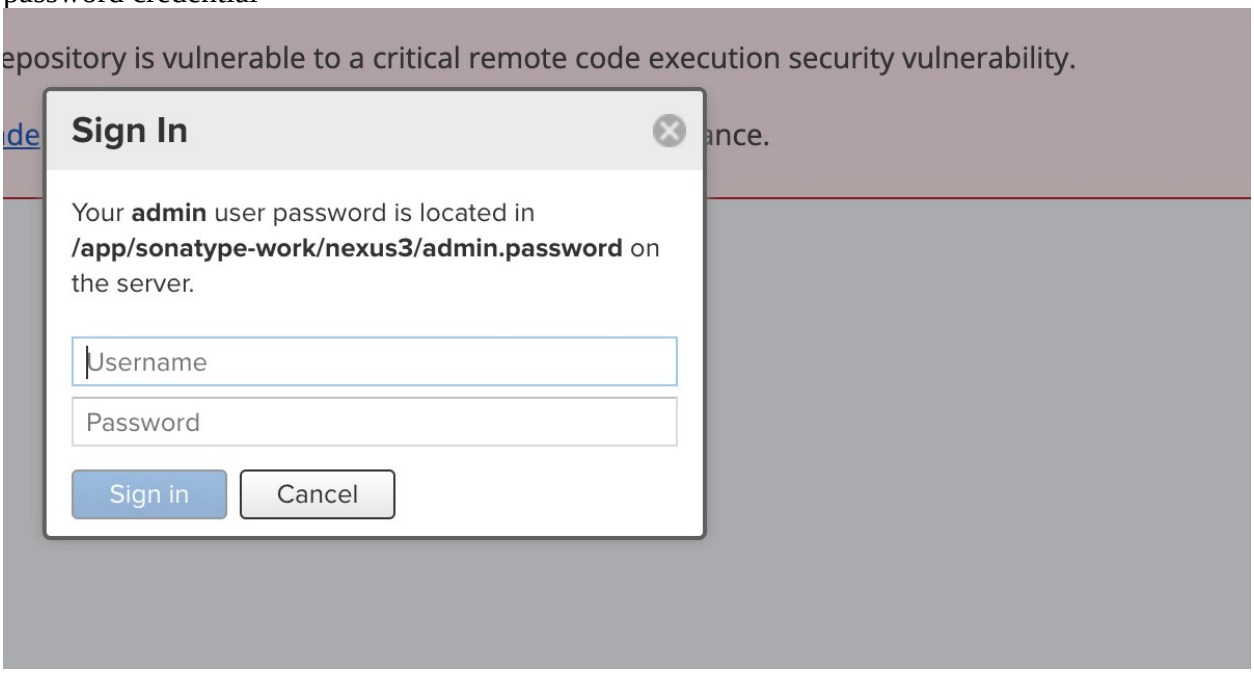
Go back to the console and pick the ip address of the nexus



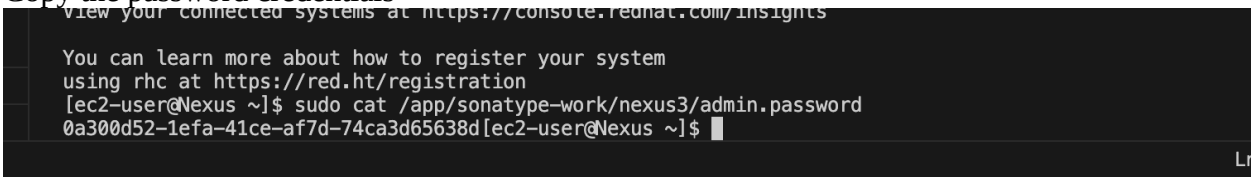
ssh into the nexus server



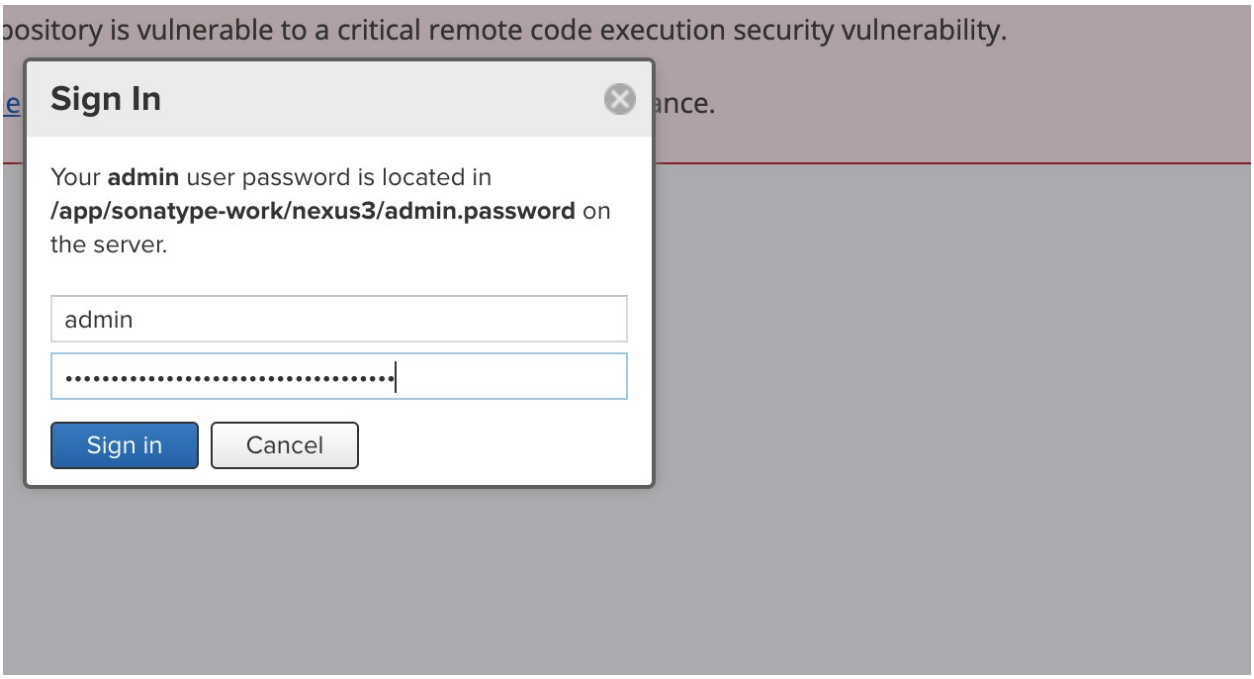
sudo cat into the nexus file path /app/sonatype-work/nexus3/admin.password to get the password credential



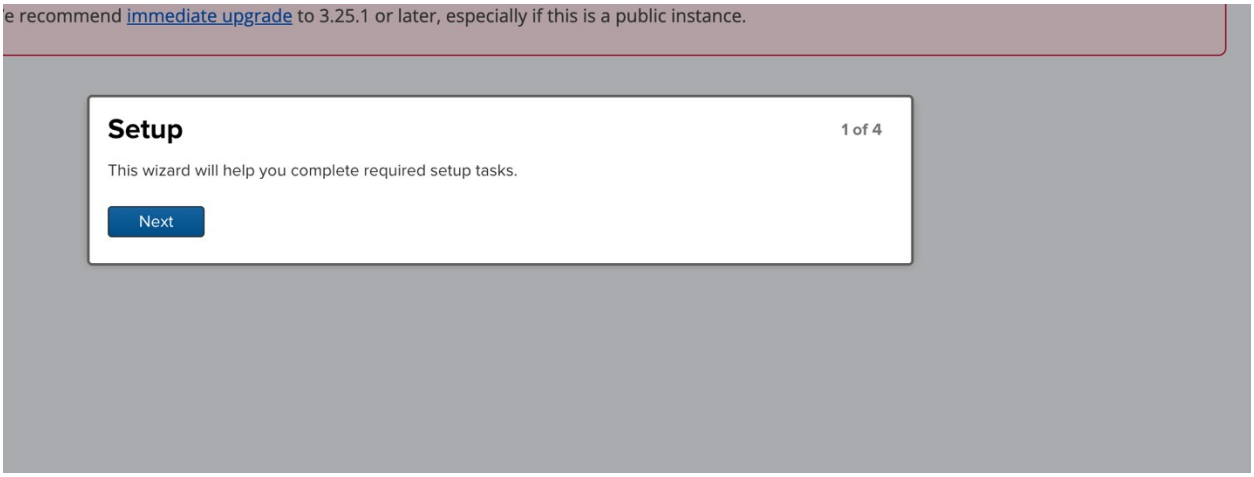
Copy the password credentials



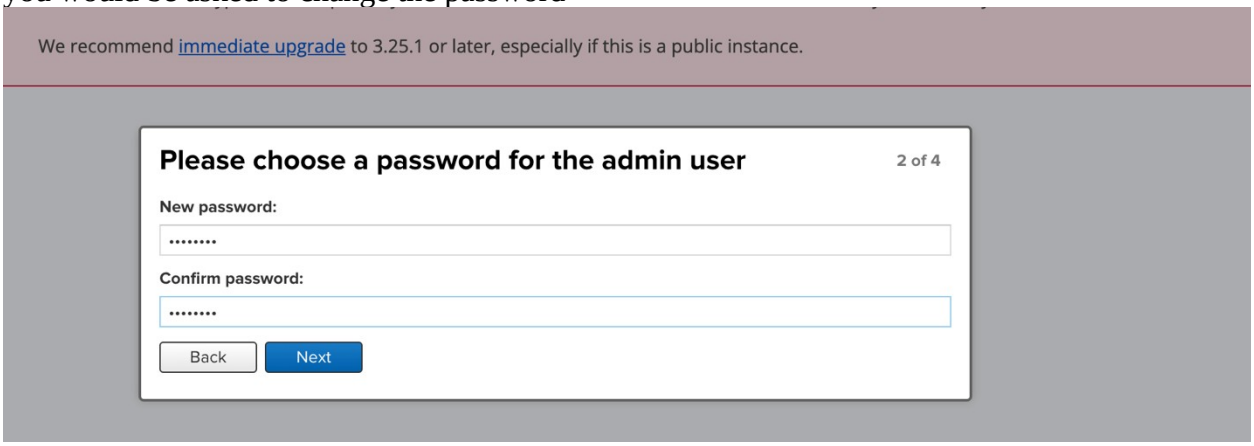
Sign in using the admin and password copied



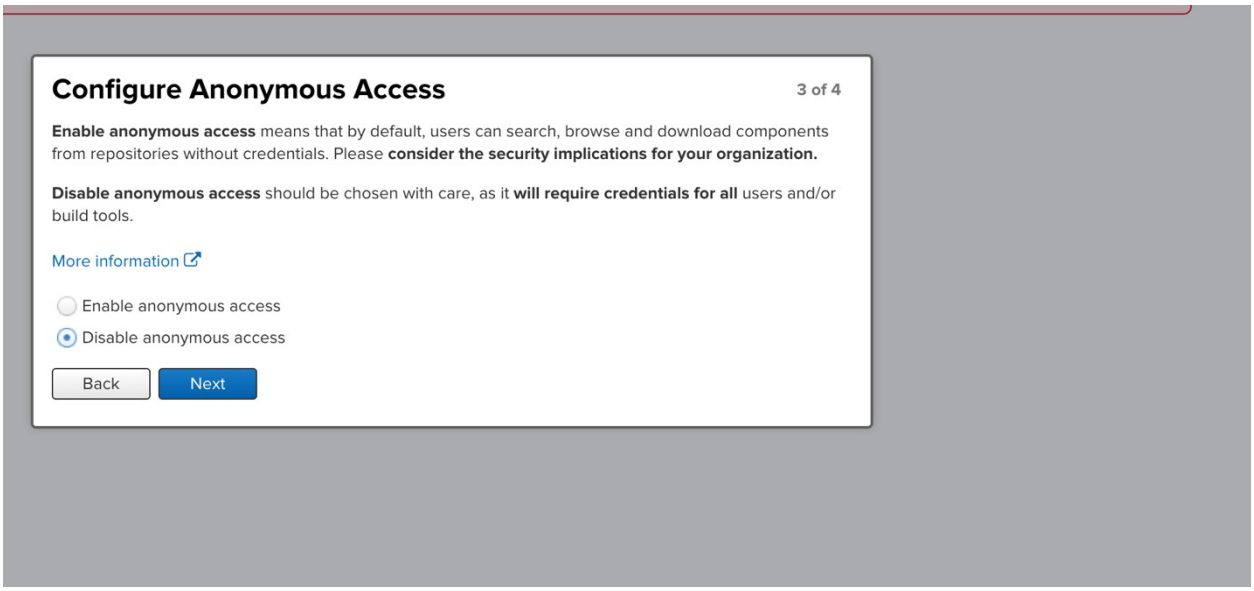
Click on next



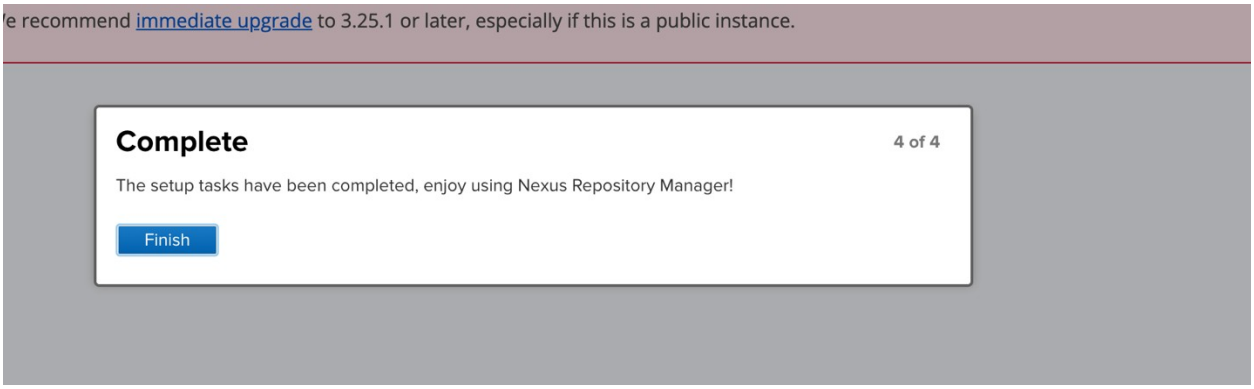
you would be asked to change the password



Disable anonymous access

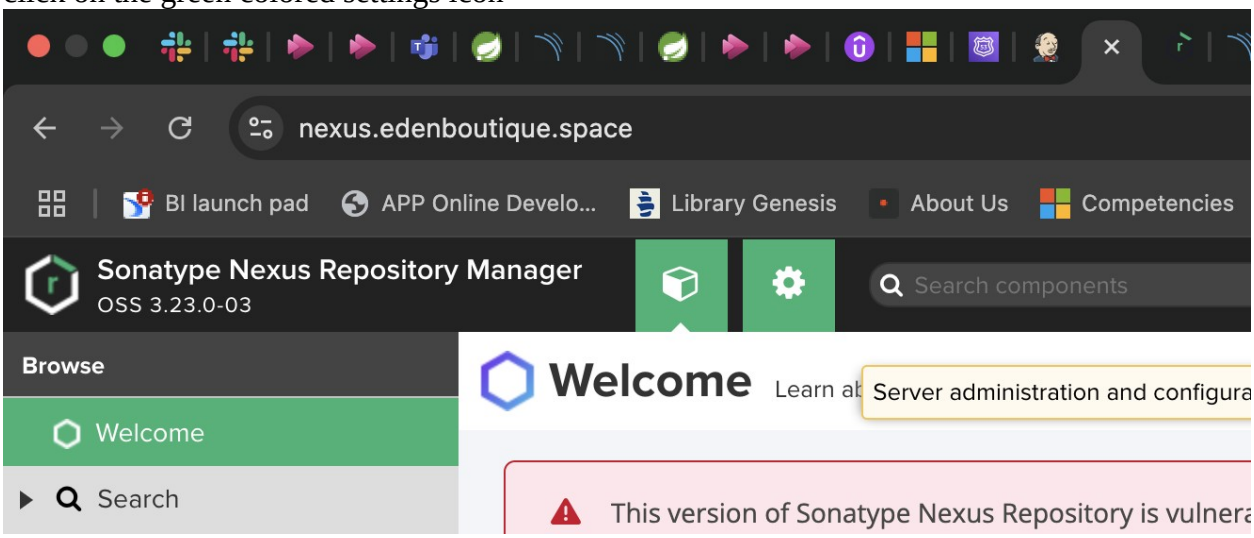


Click on finish

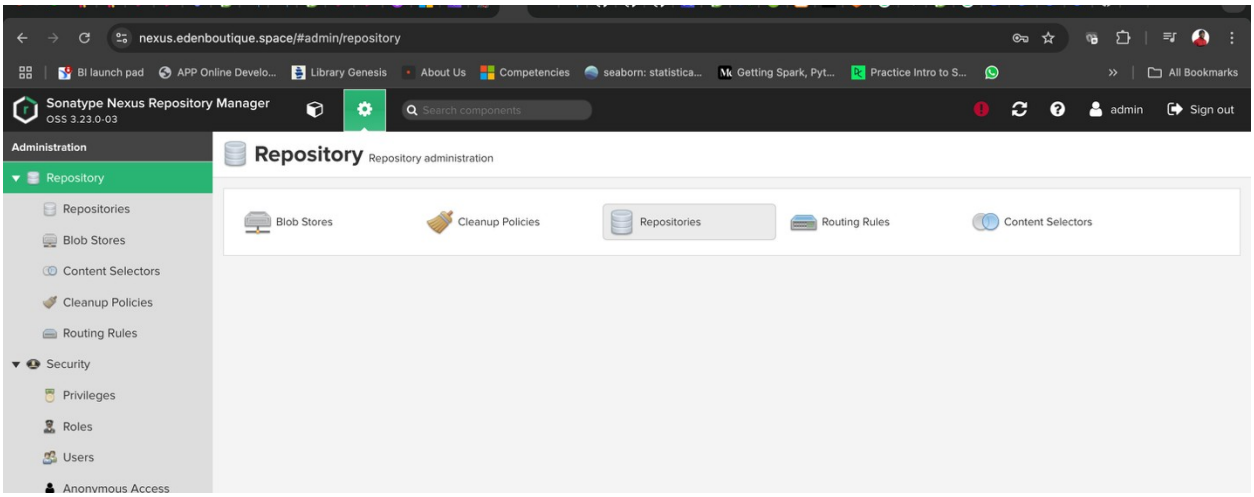


Now is time to create the repository, we would be creating the maven and docker hosted repository.

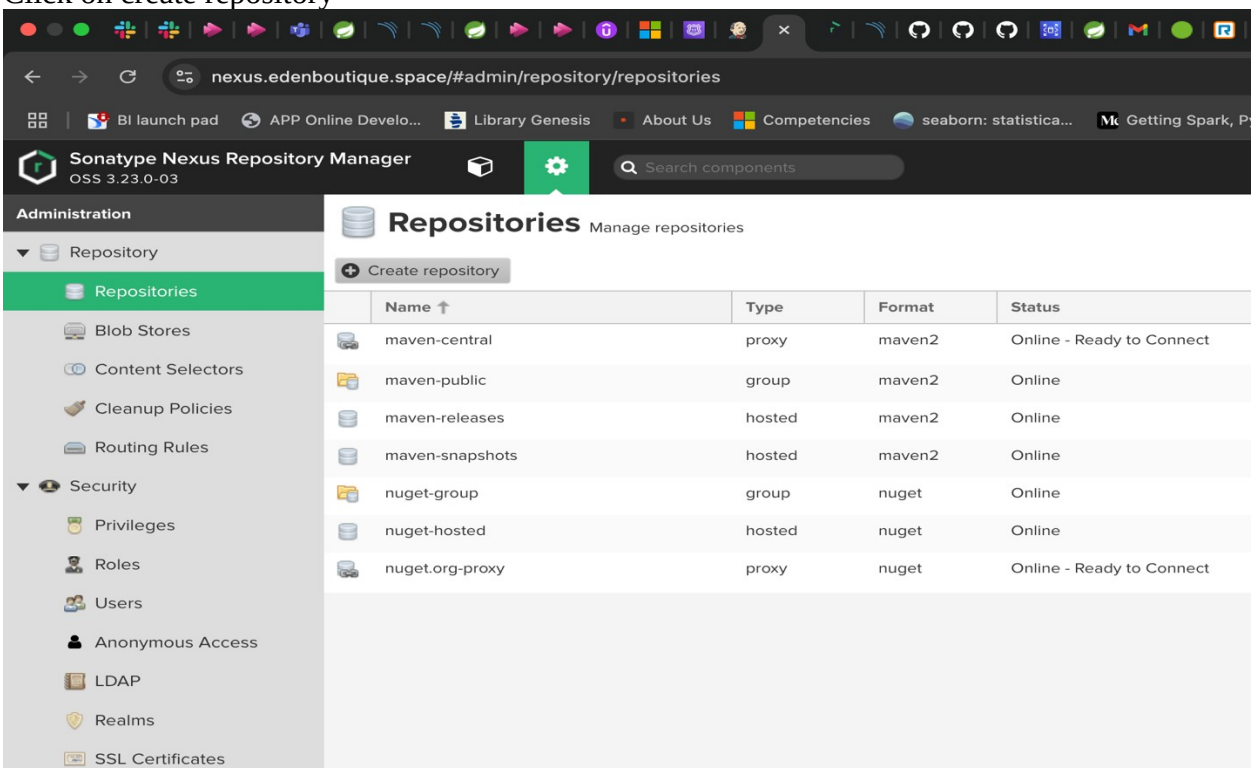
click on the green colored settings icon



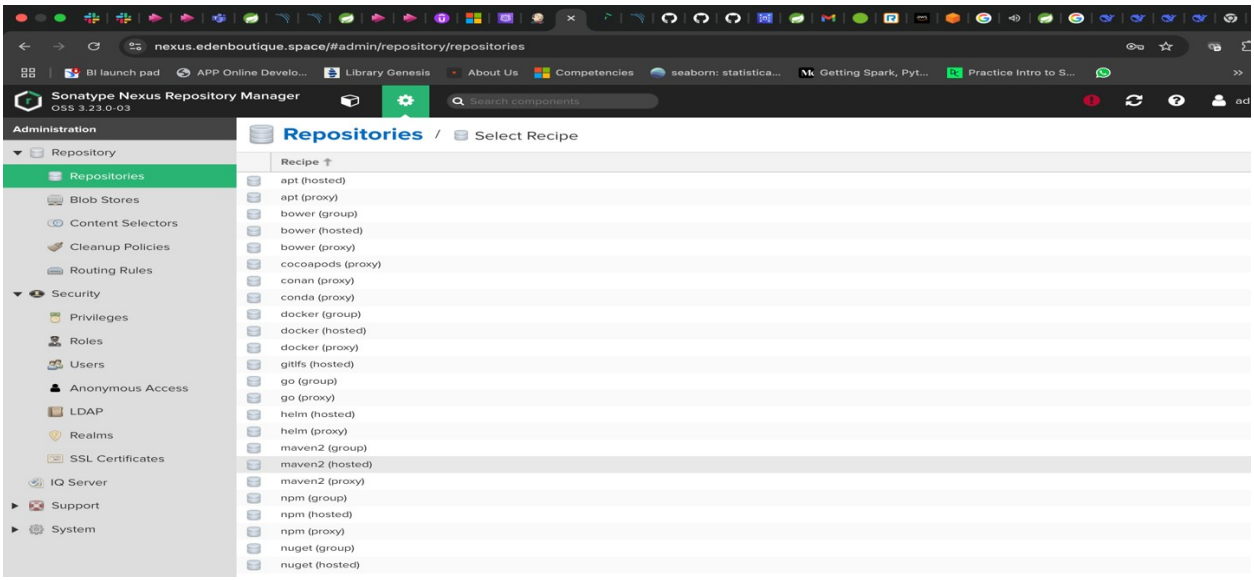
click on repositories



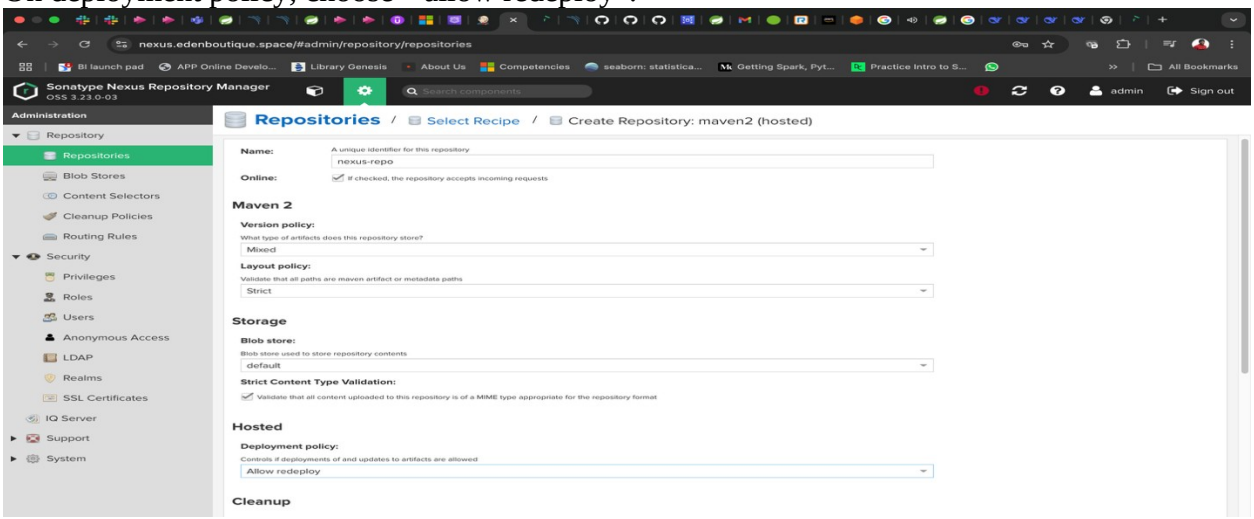
Click on create repository



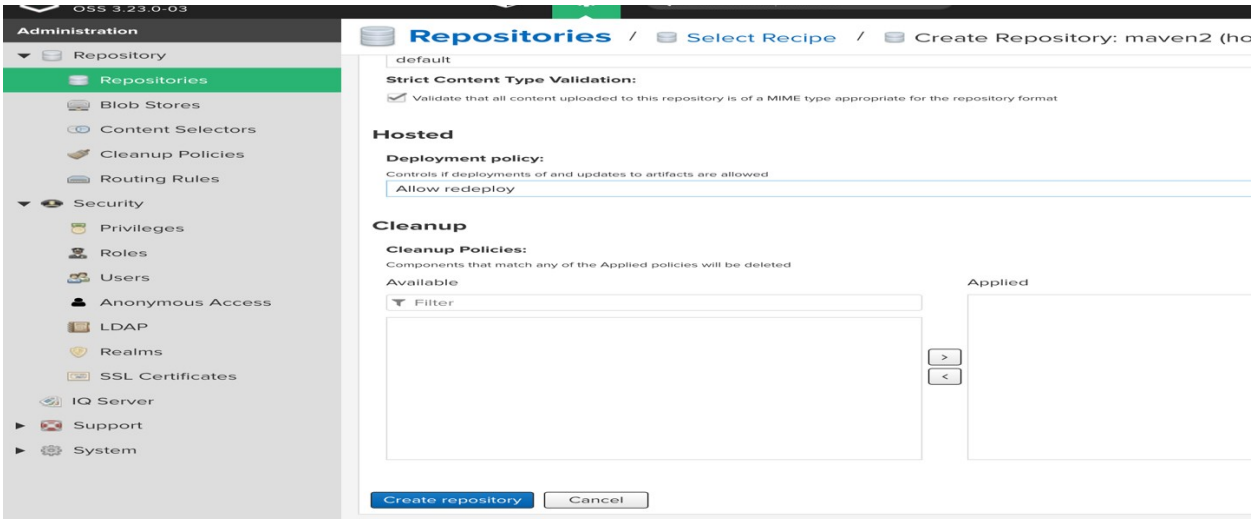
Search for maven2 hosted repository



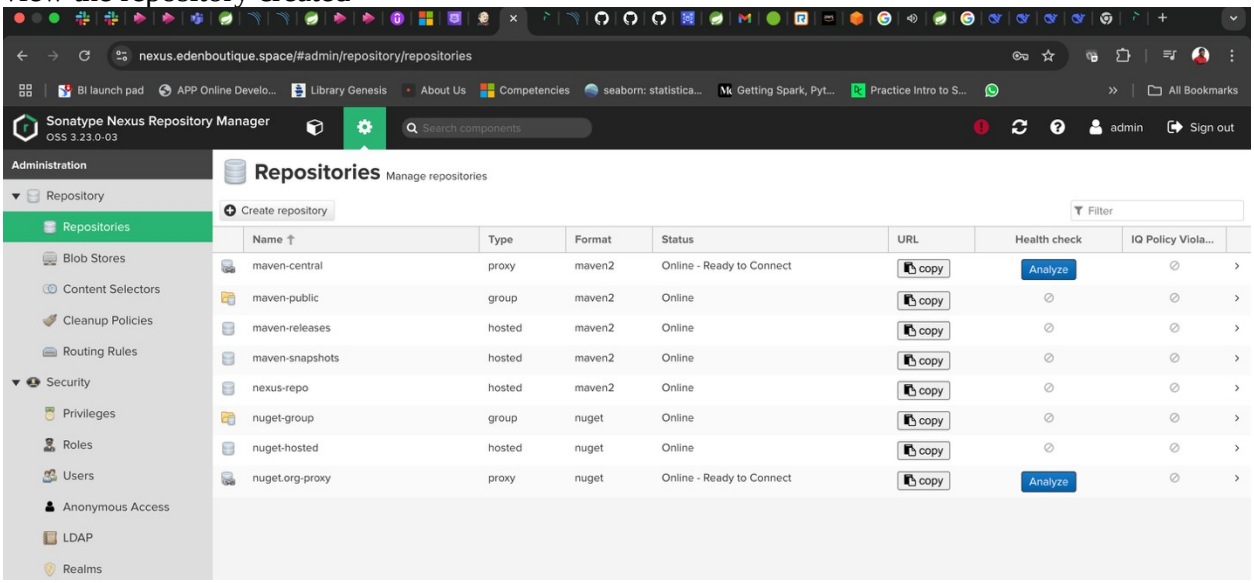
Give your repository a name.
On version policy, choose “mixed”.
On deployment policy, choose “allow redeploy”.



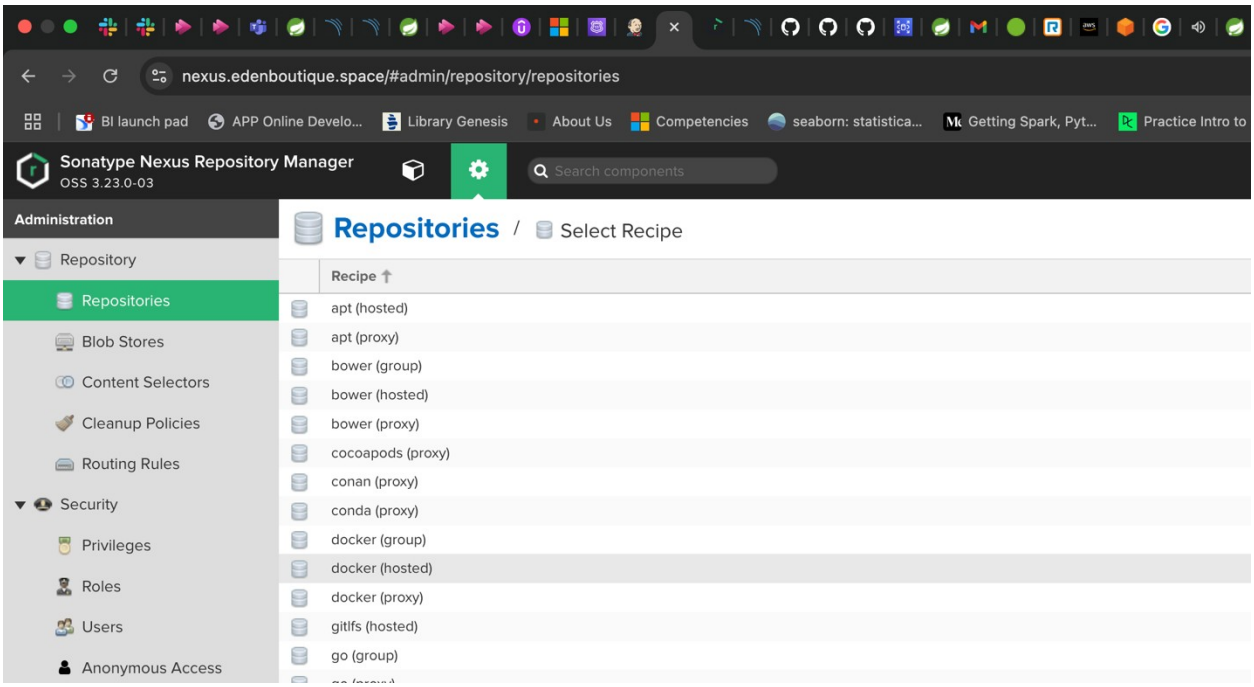
then click on create repository



view the repository created



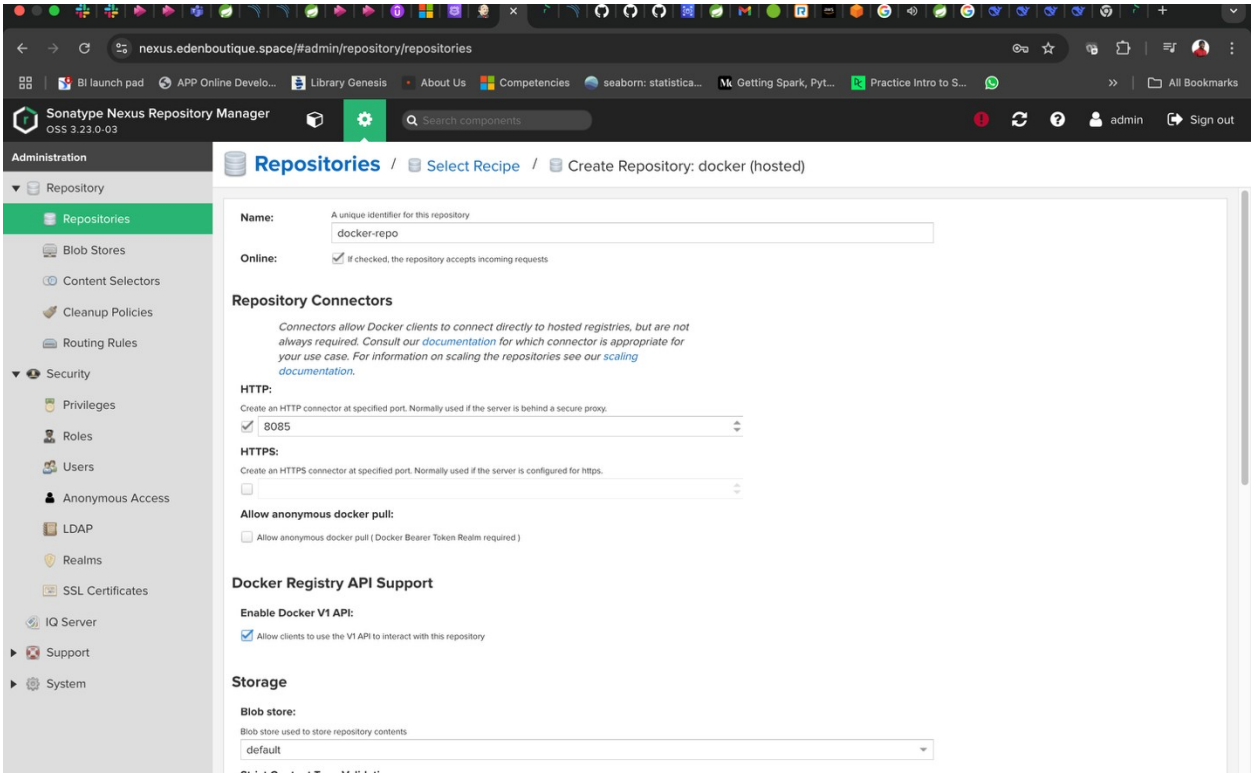
Next, search for docker hosted repository



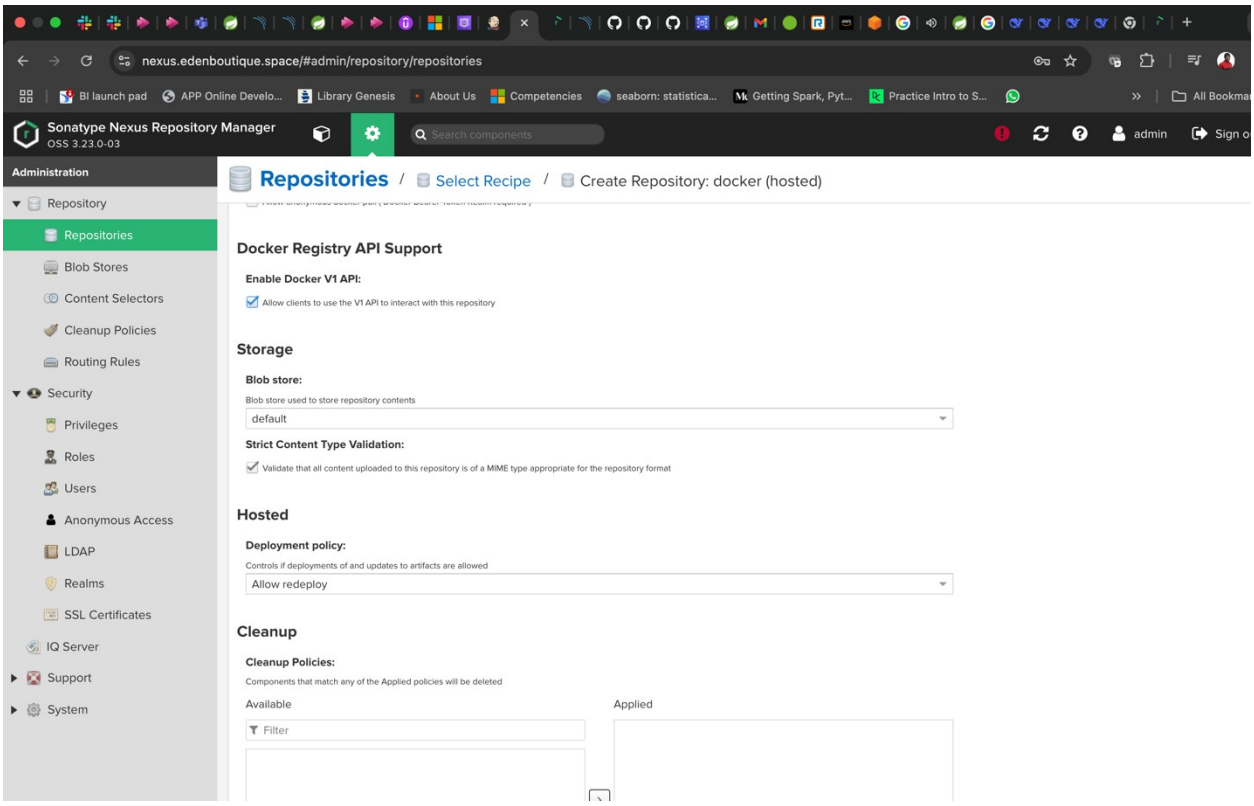
Give it a name.

Create an HTTP connector on port 8085 this is where the port where the docker image would be exposed

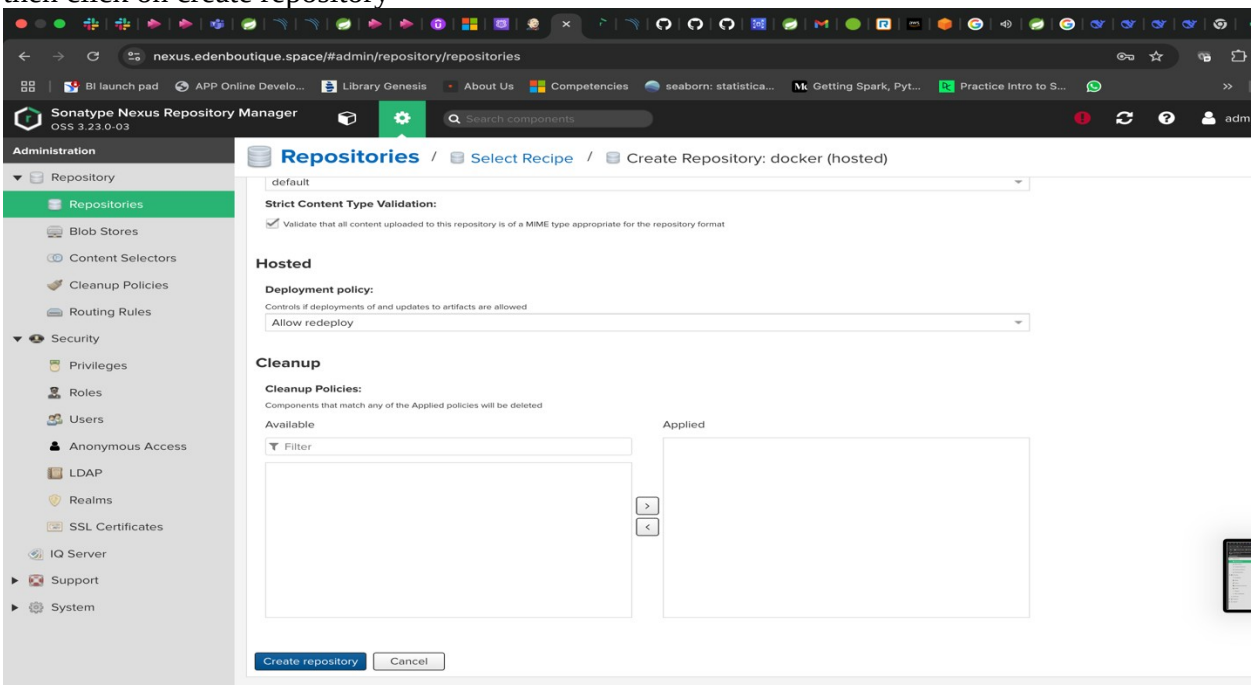
then enable docker vi API



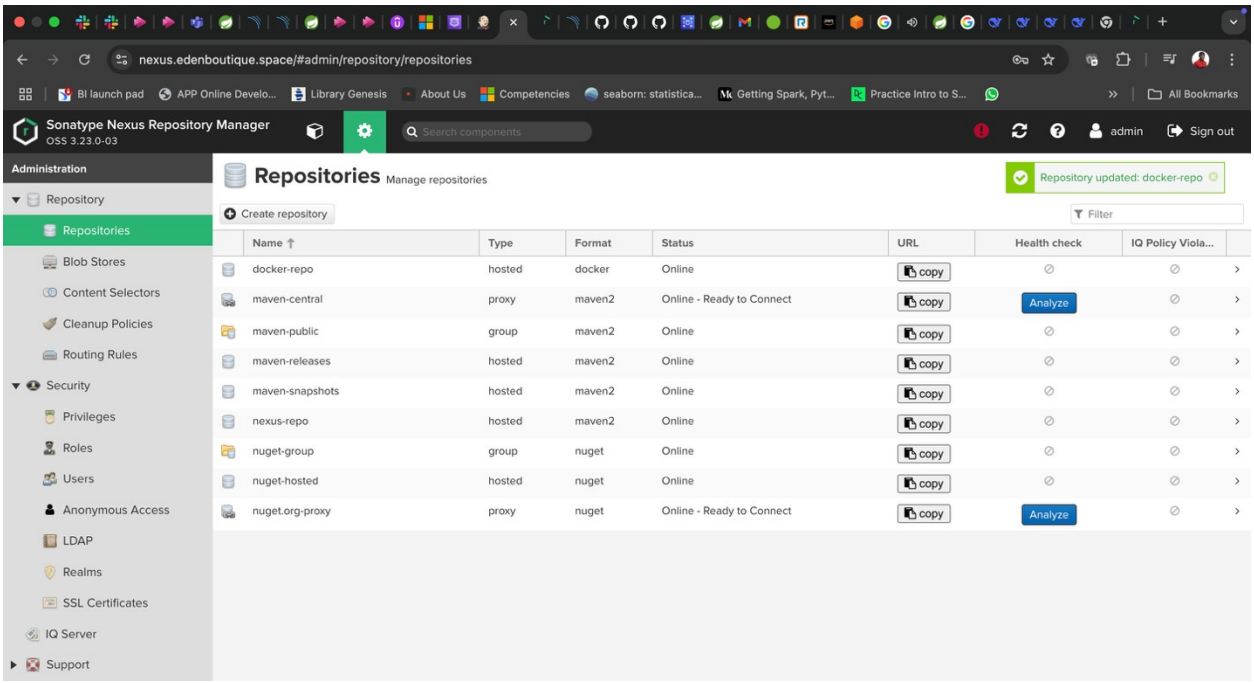
Allow redeploy on deployment policy



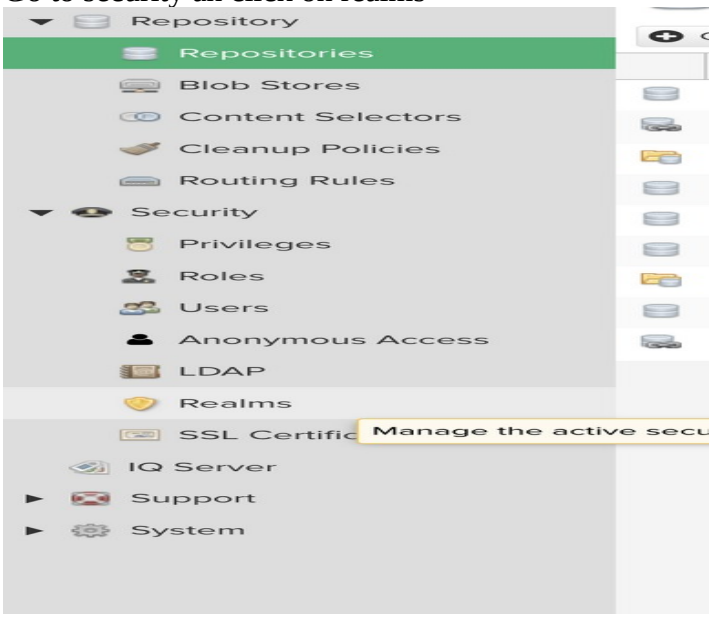
then click on create repository



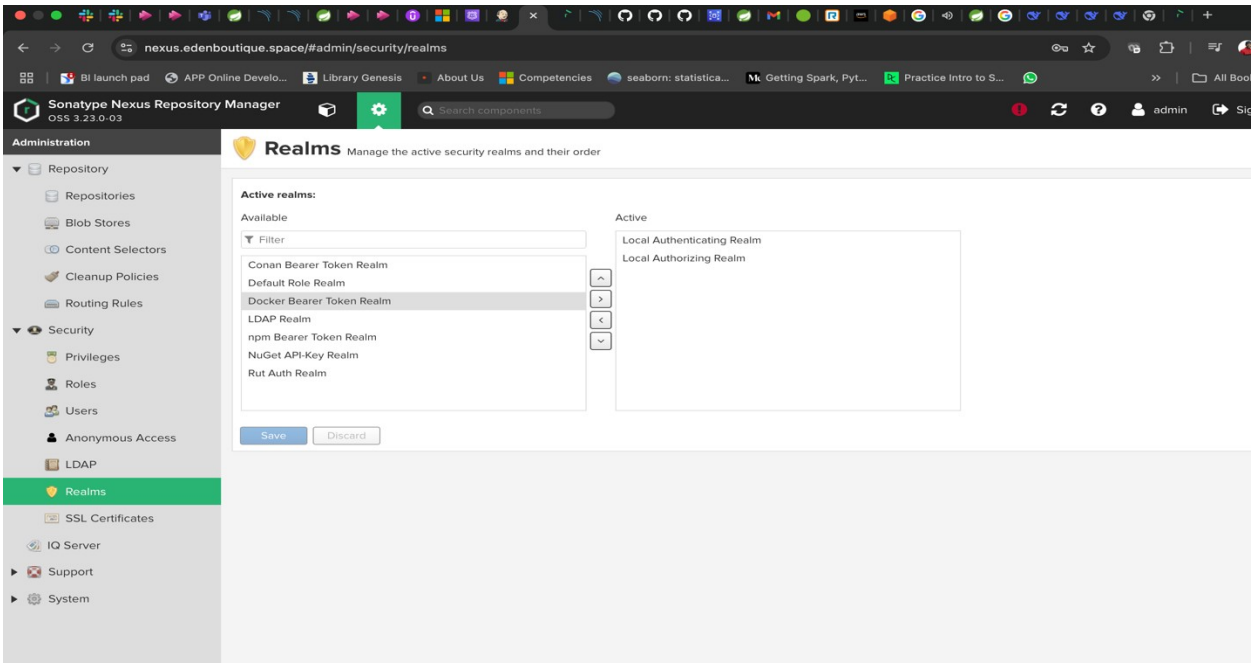
view the docker repo created



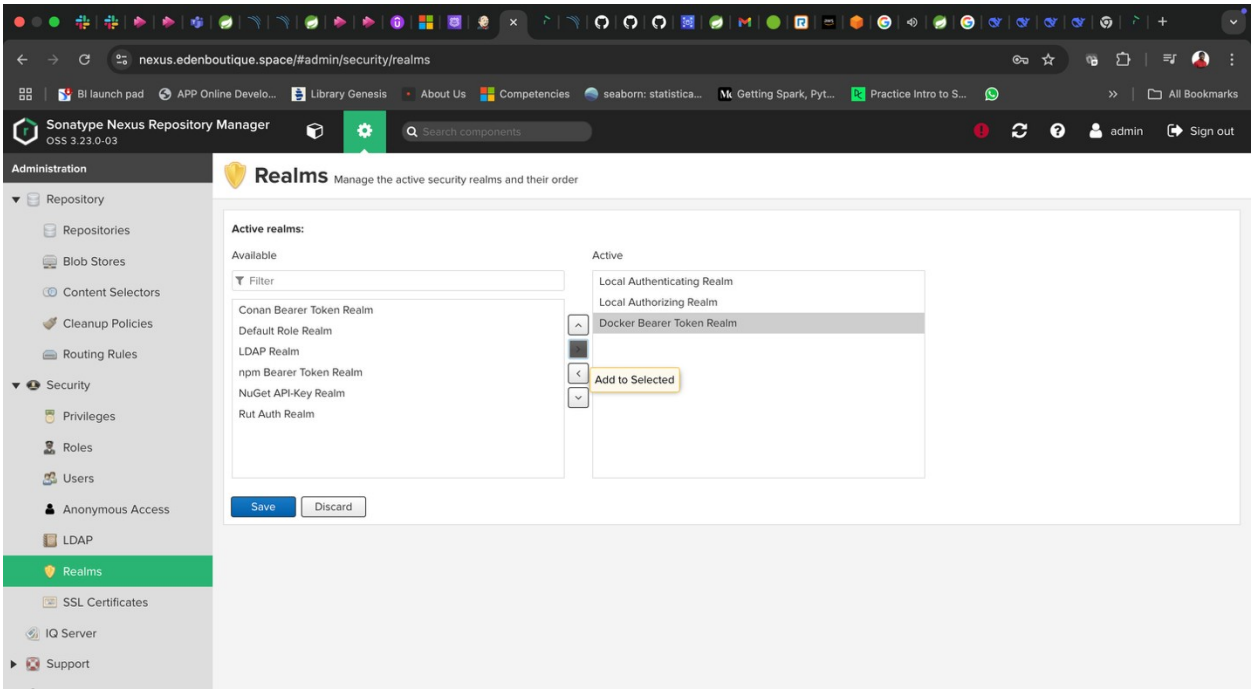
Go to security an click on realms



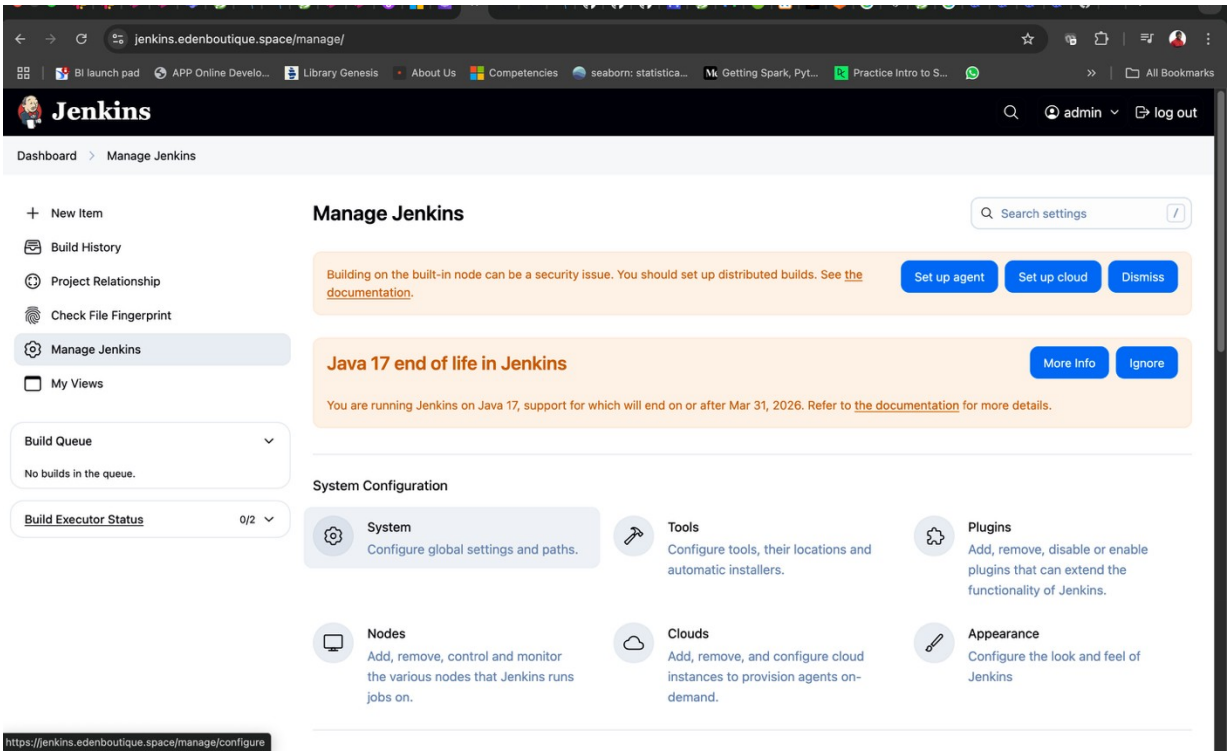
click on “docker bearer token realms”



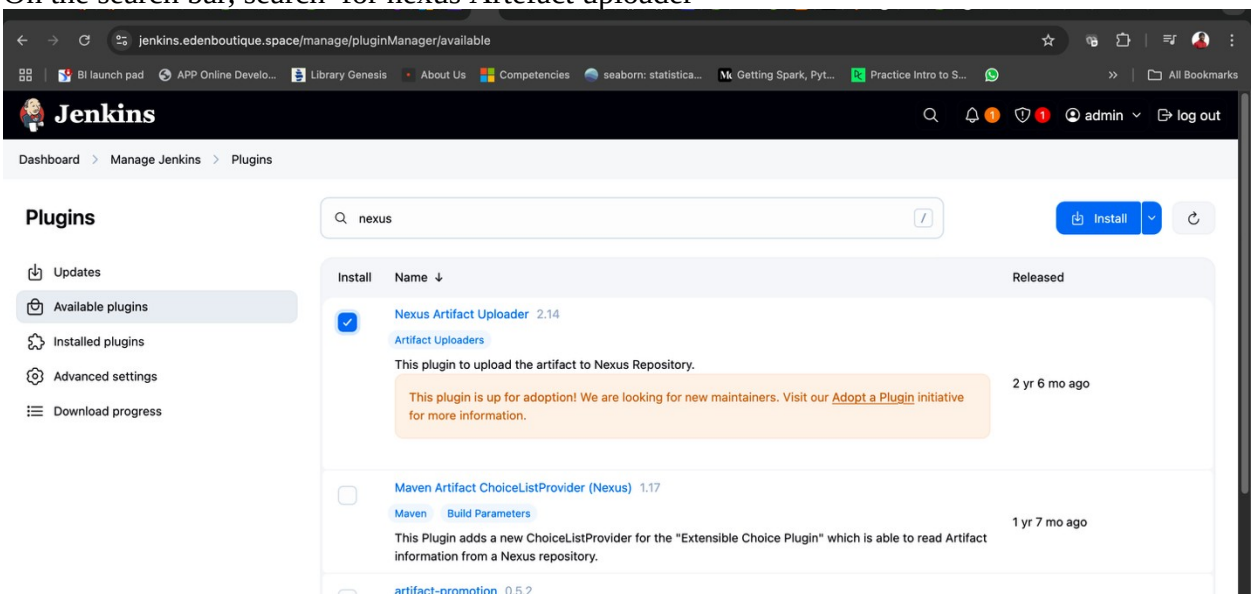
Click on the docker bearer token realms and add to the active side



Now install nexus artefact uploader.
Go to “manage jenkins”
click on systems



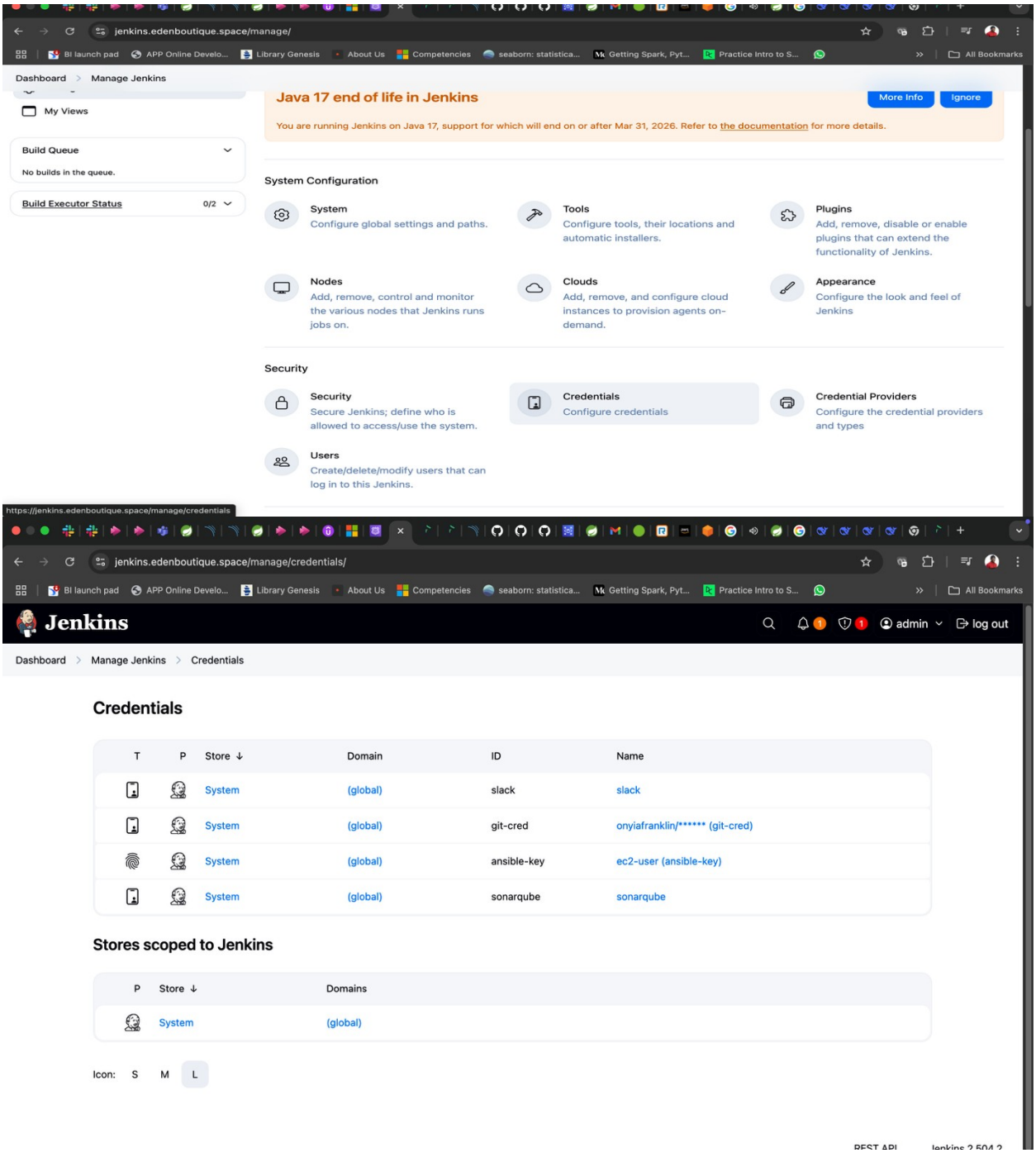
On the search bar, search for nexus Artefact uploader



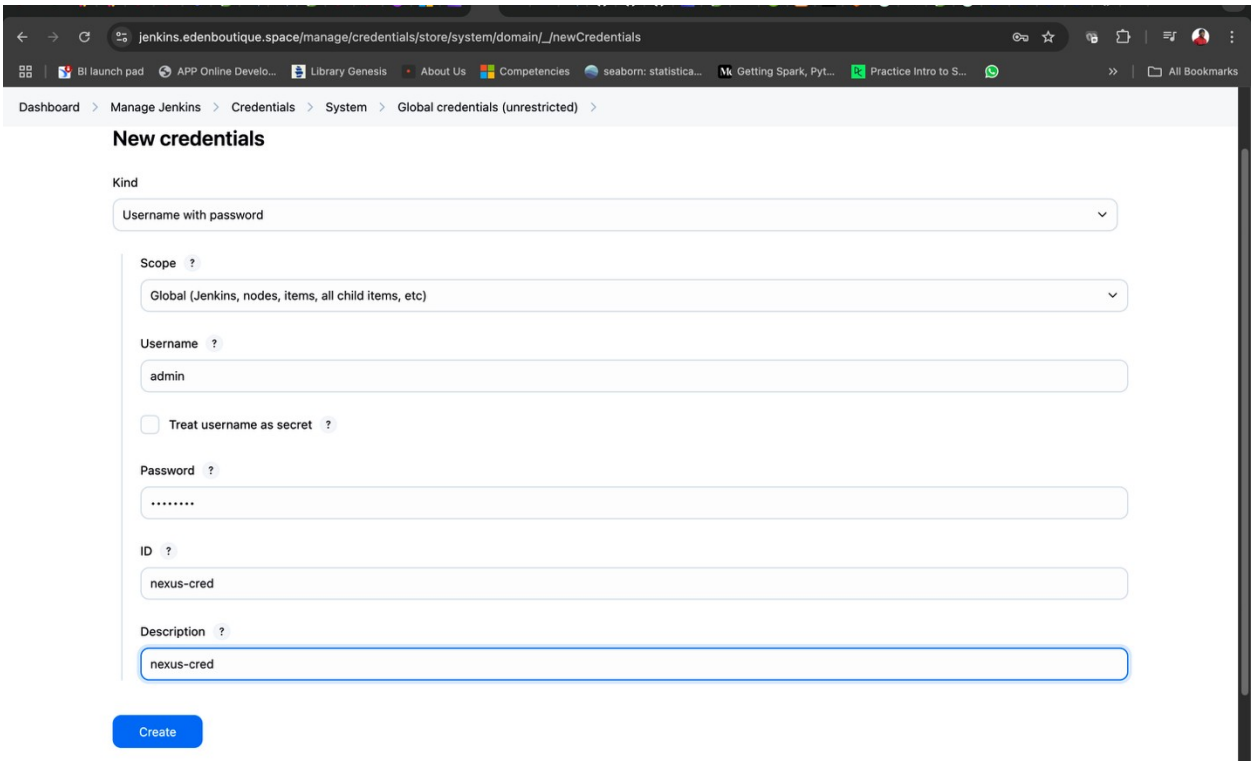
Now lets add the nexus credentials where artefact uploader would upload the artefact from maven.

GO TO MANAGE JENKINS

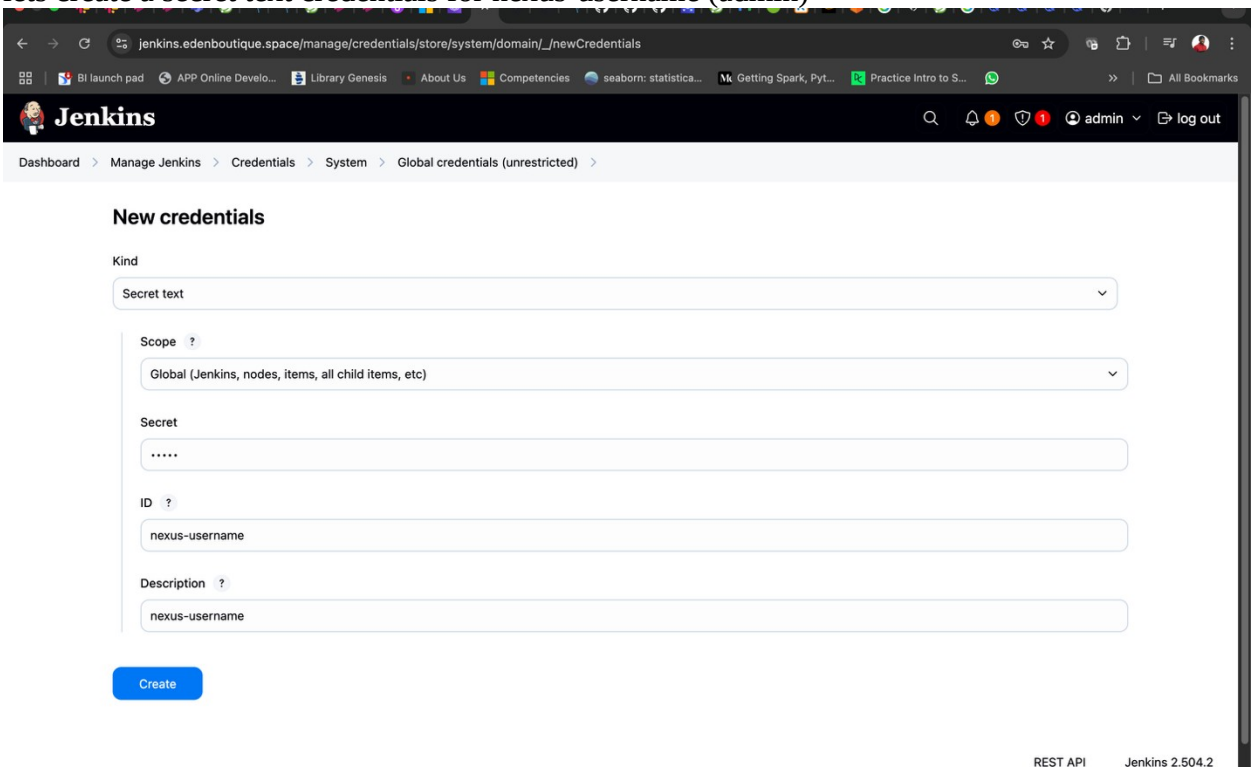
Click on credentials



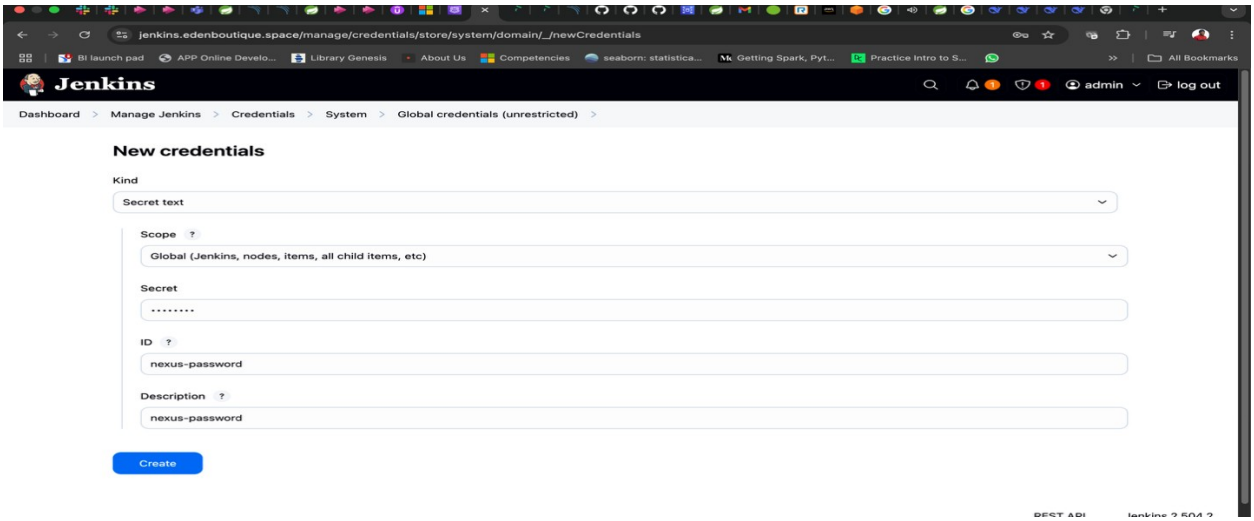
Click on global to add credentials.
choose username and password as kind of credentials and add the credentials



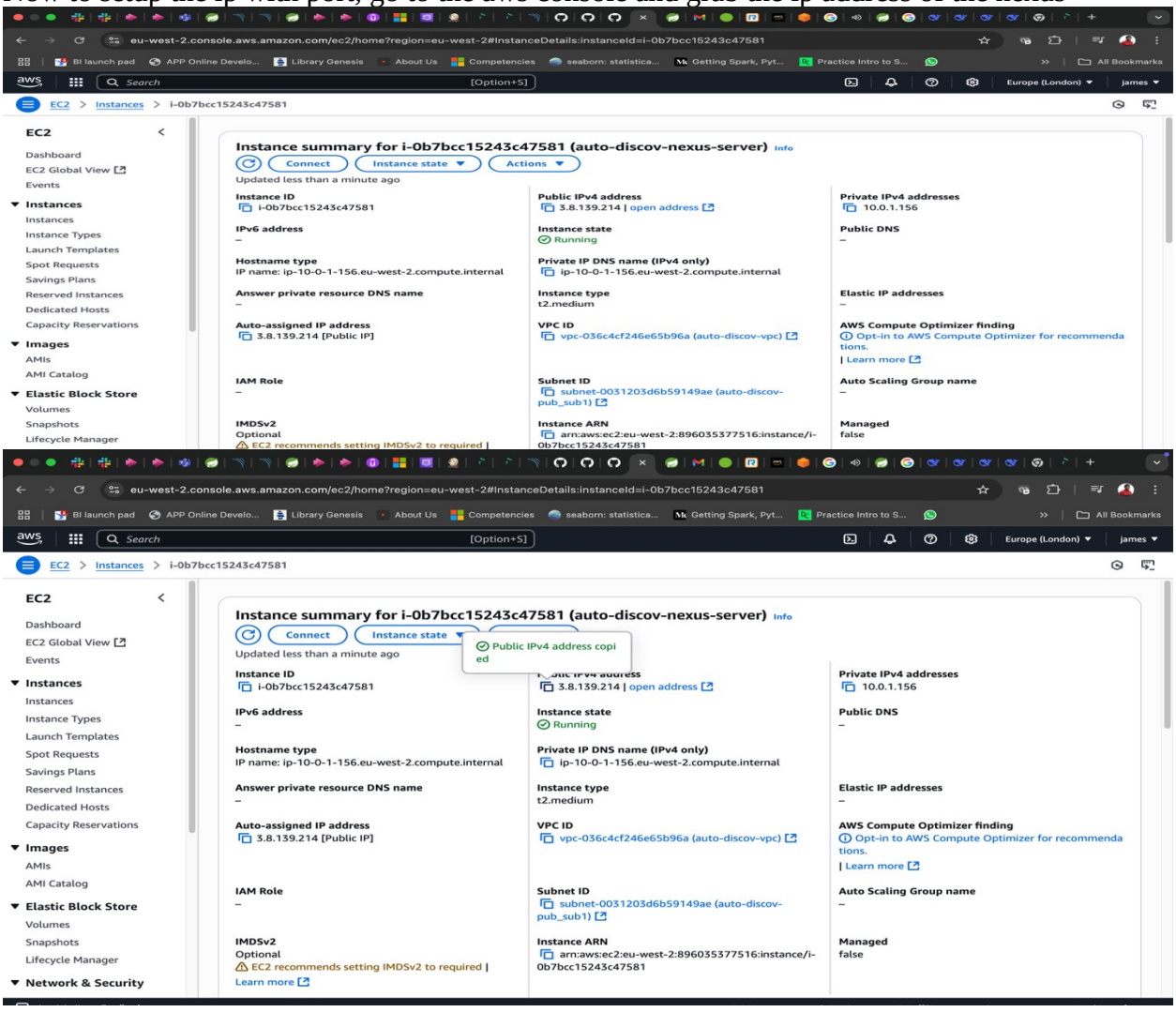
Next we need to setup the credentials that would be used to pull the docker image from the nexus repository opened on port 8085
to set it up we need to set up the username , password and (ip with port 8085)
lets create a secret text credentials for nexus-username (admin)



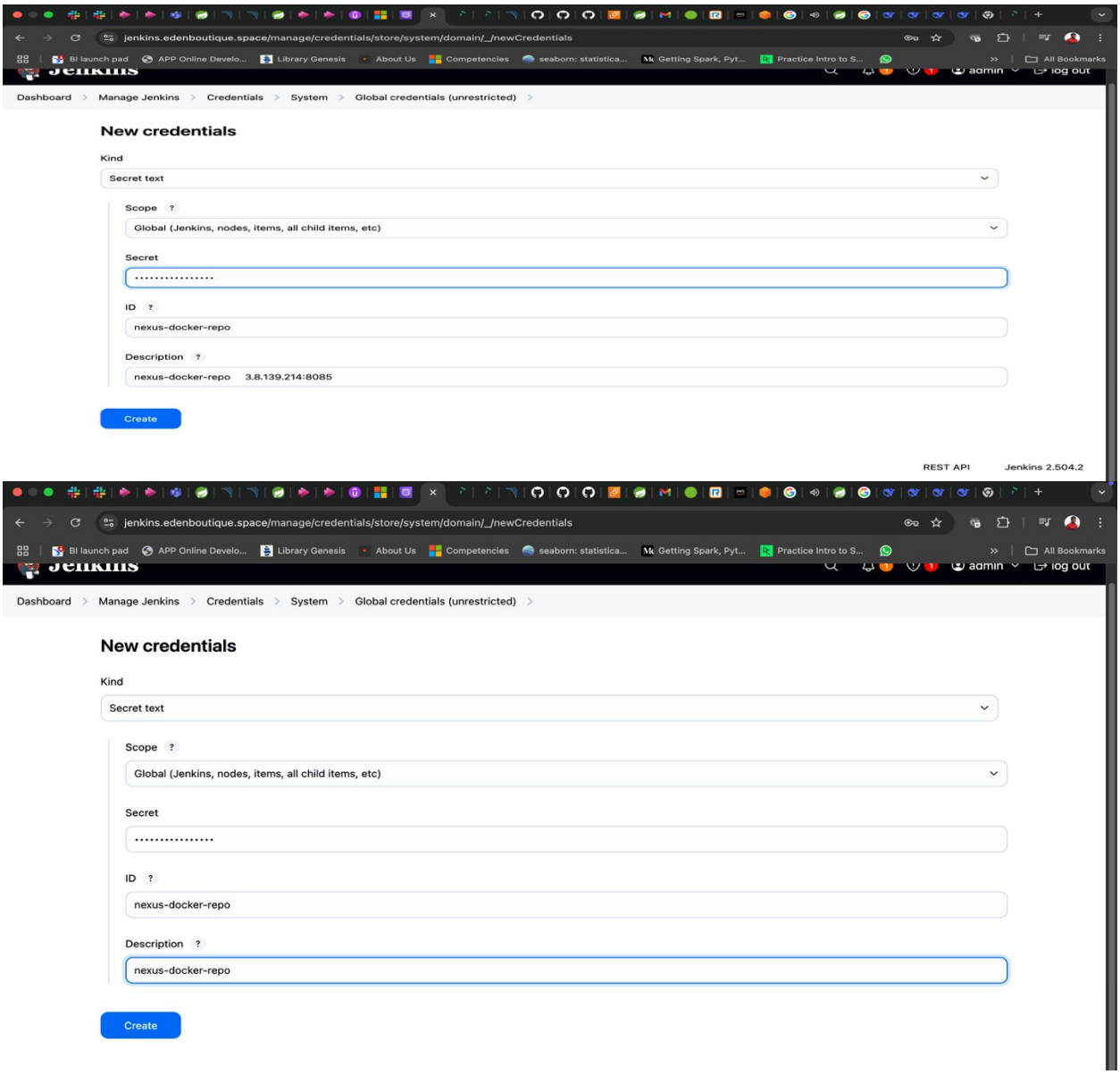
create secret text nexus-password credentials (admin123)



Now to setup the ip with port, go to the aws console and grab the ip address of the nexus

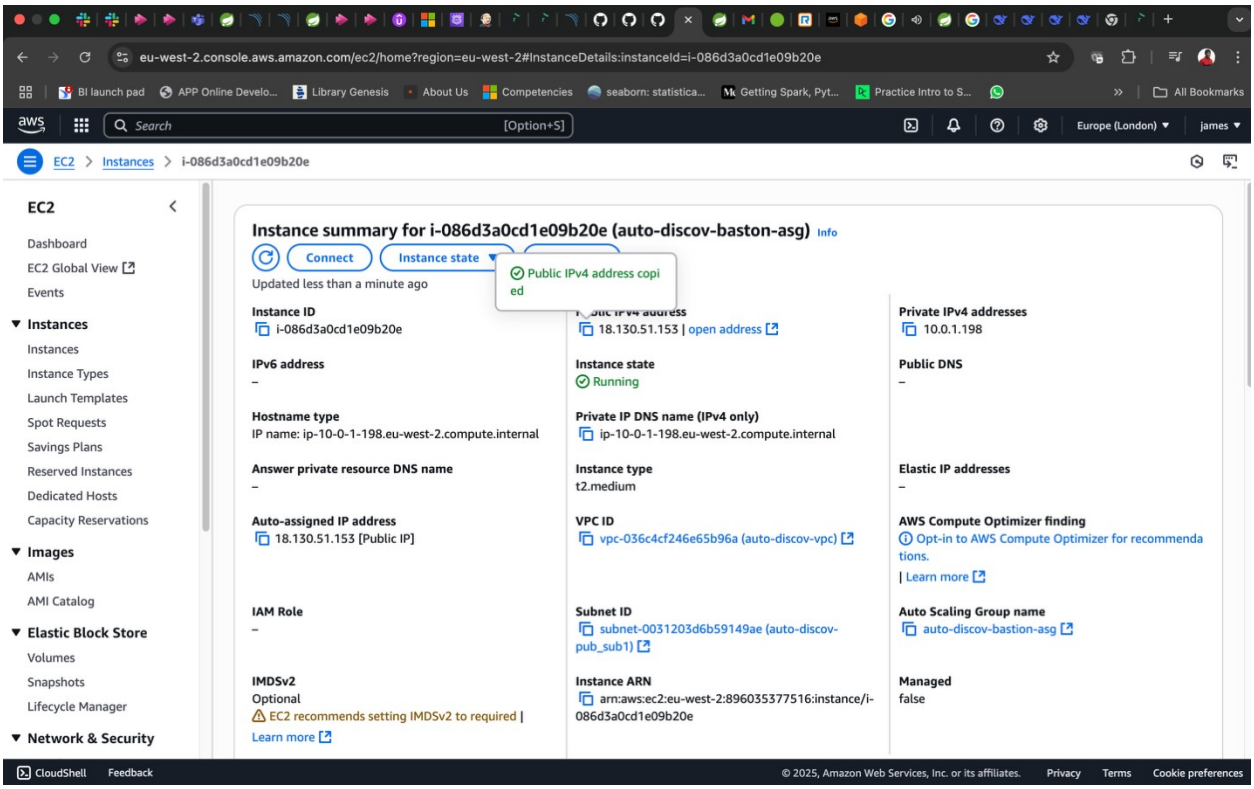


create asecret text credential with the ip and port of the nexus docker repo in the format (ipaddress:8085)

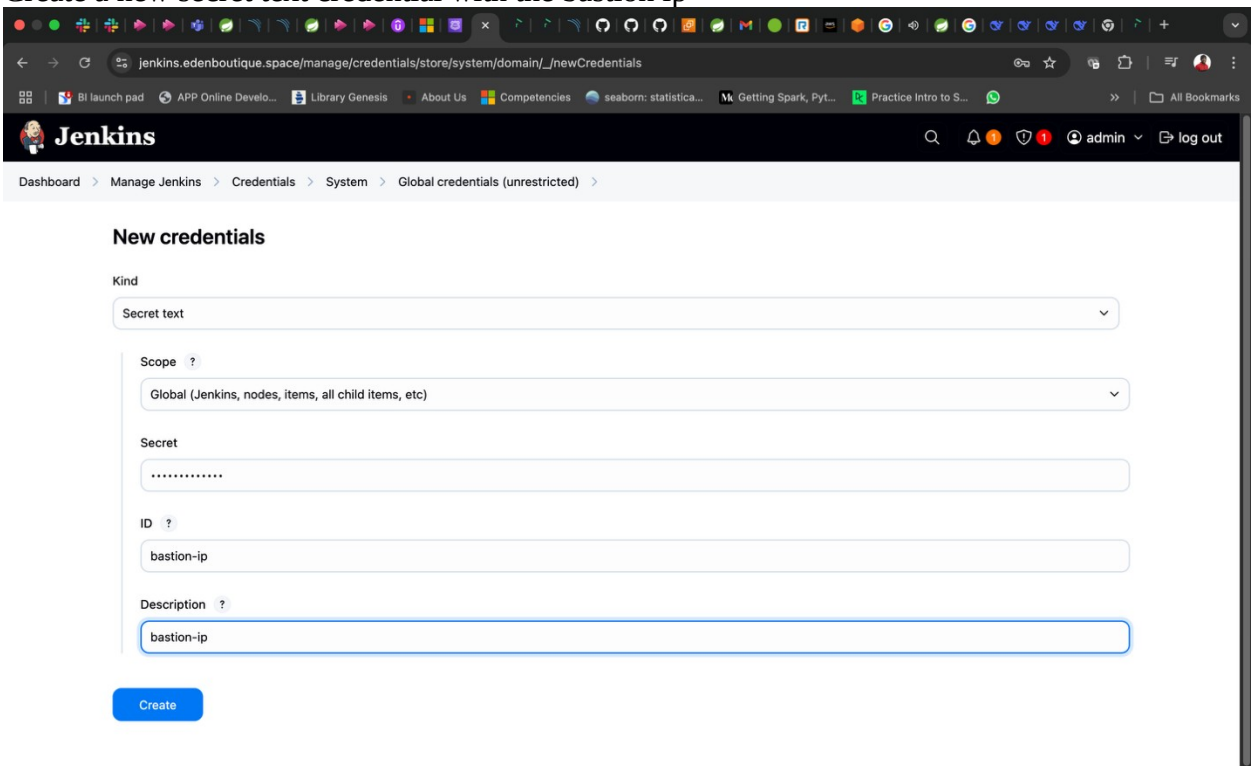


Next we need to create credential for the bastion ip so that Jenkins would use the credentials to logging to stage and prod server through tunnelling

copy the bastion host public ip from the console

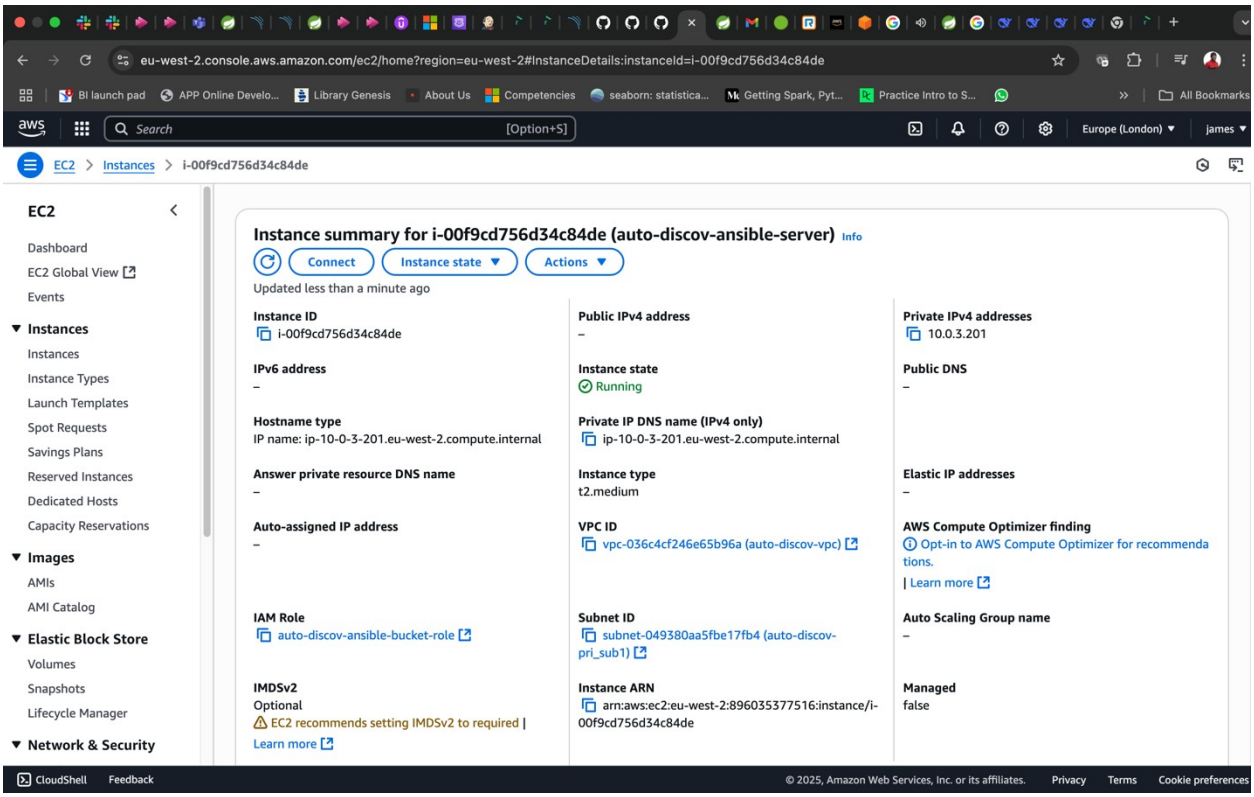


Create a new secret text credential with the bastion ip

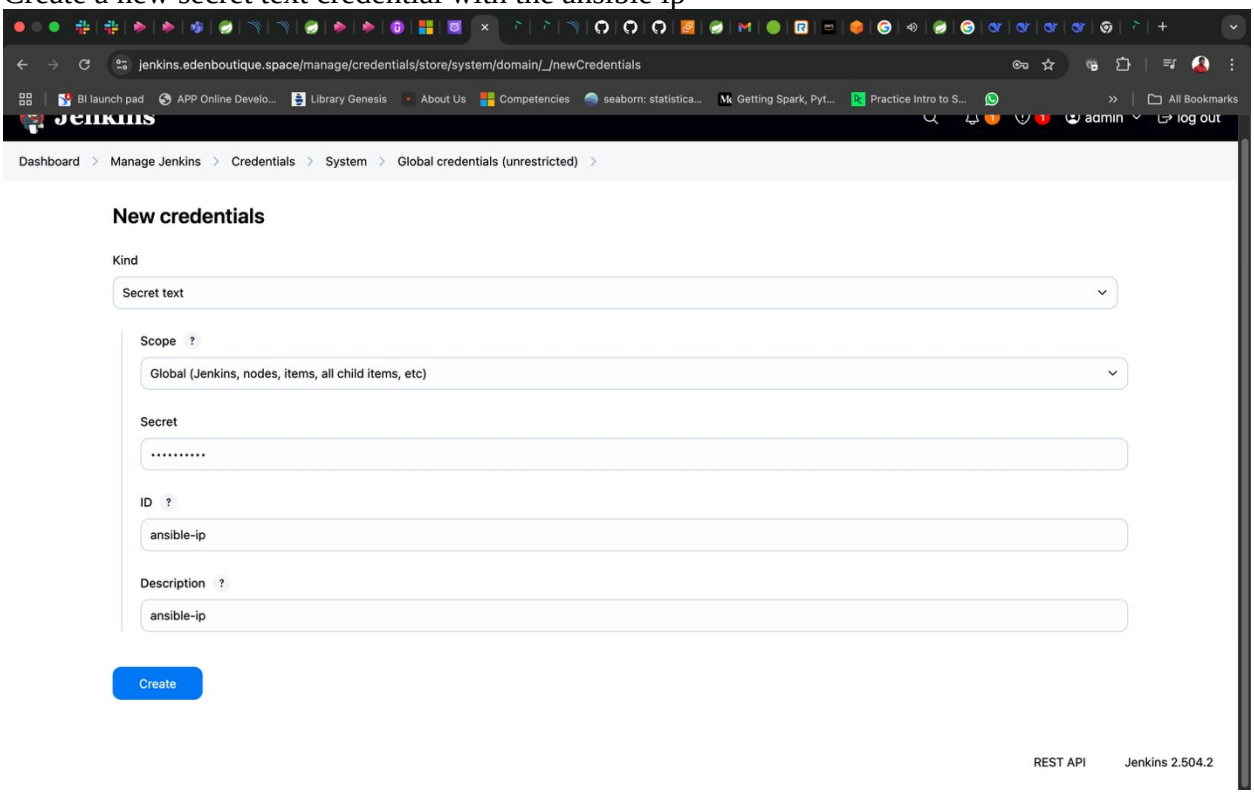


Next we need to create credential for the ansible ip so that Jenkins would use the credentials to logging to stage and prod server through tunnelling

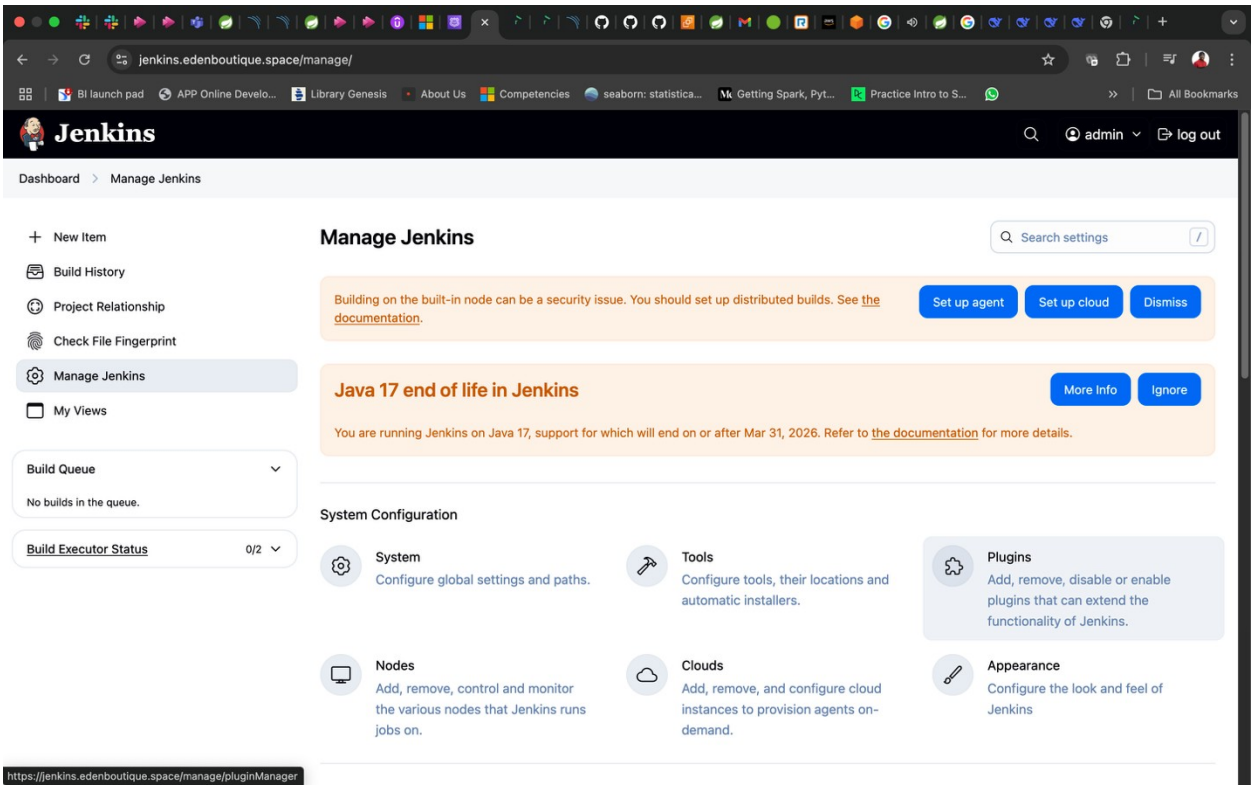
copy the ansible host private ip from the console



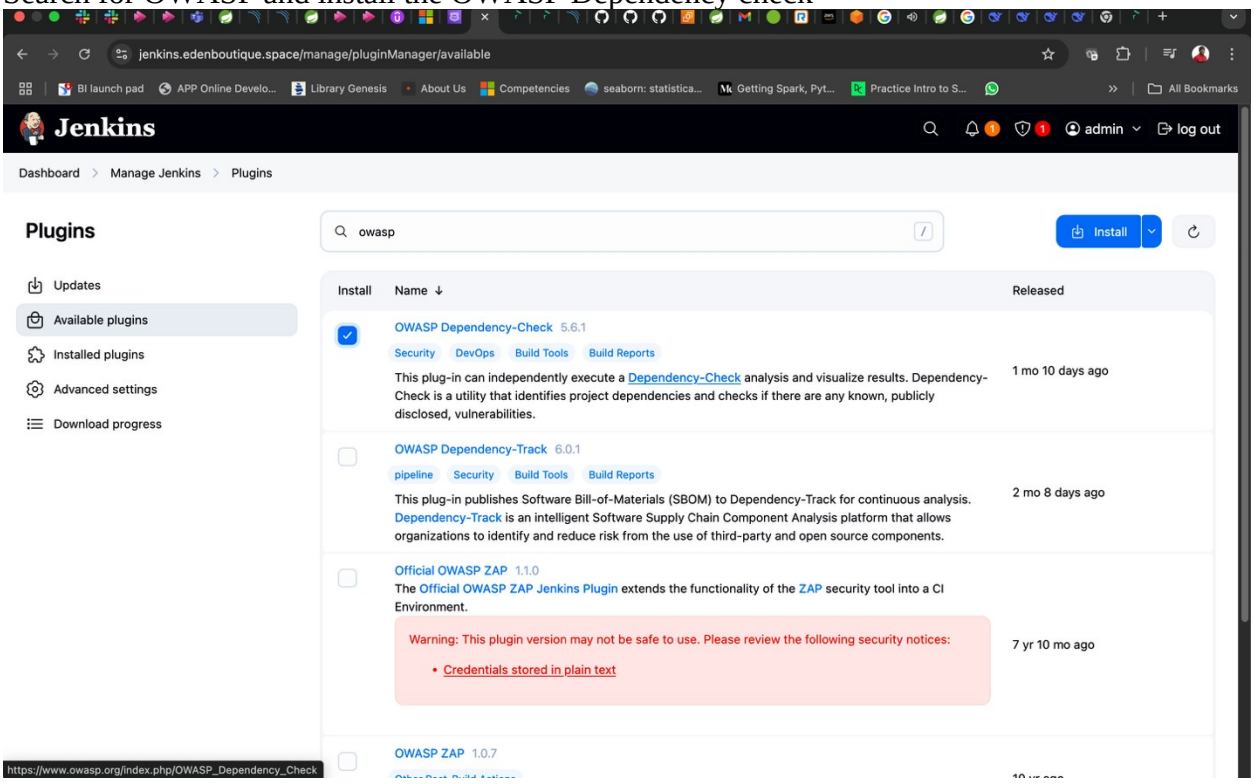
Create a new secret text credential with the ansible ip



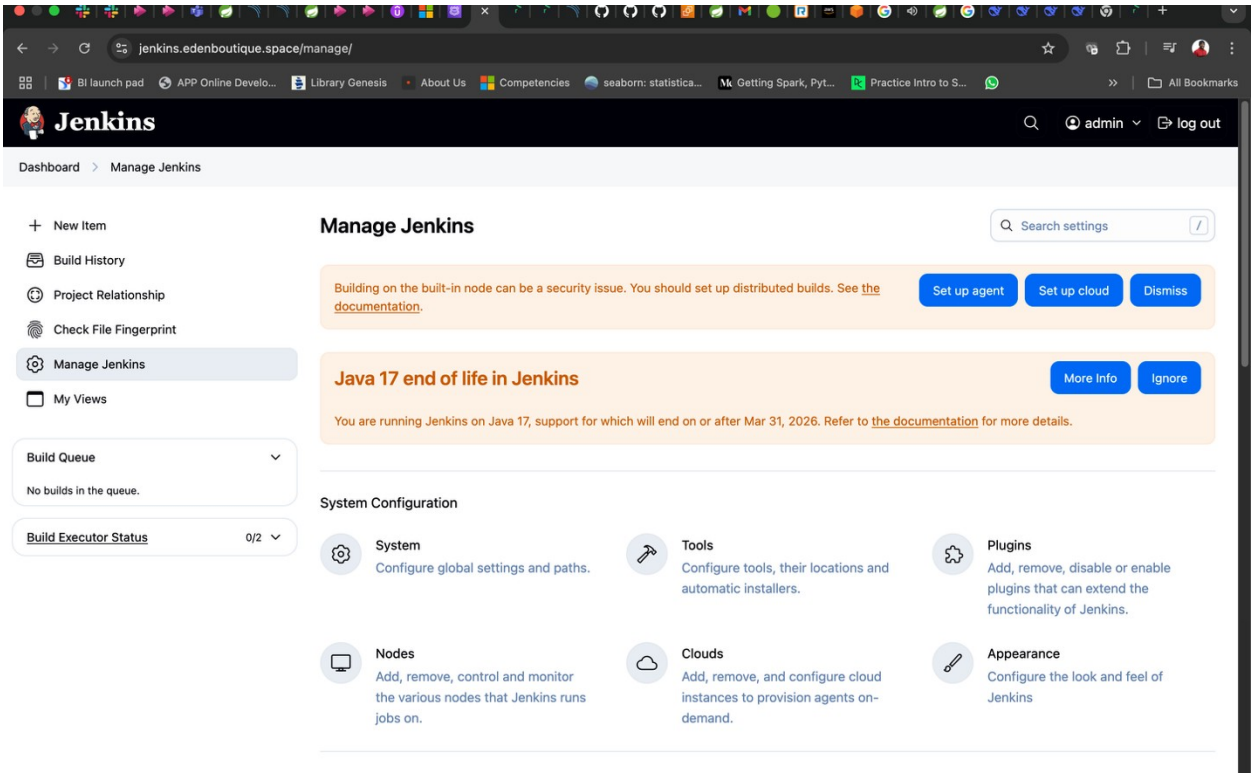
Now we need to add a plugin for OWASP dependency scan and also create nvd key that it would use for faster scanning.
Go to manage jenkins and click on credentials



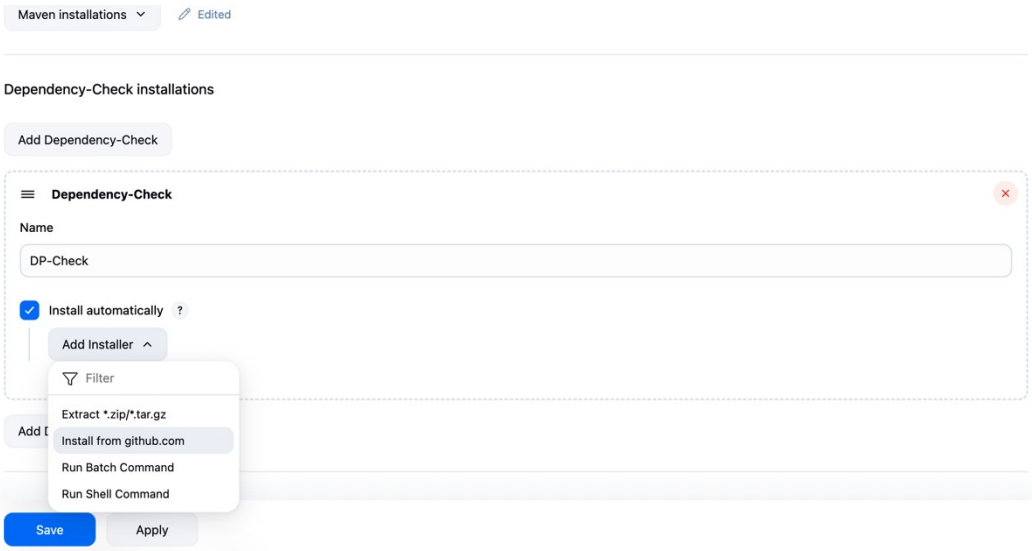
Search for OWASP and install the OWASP Dependency check



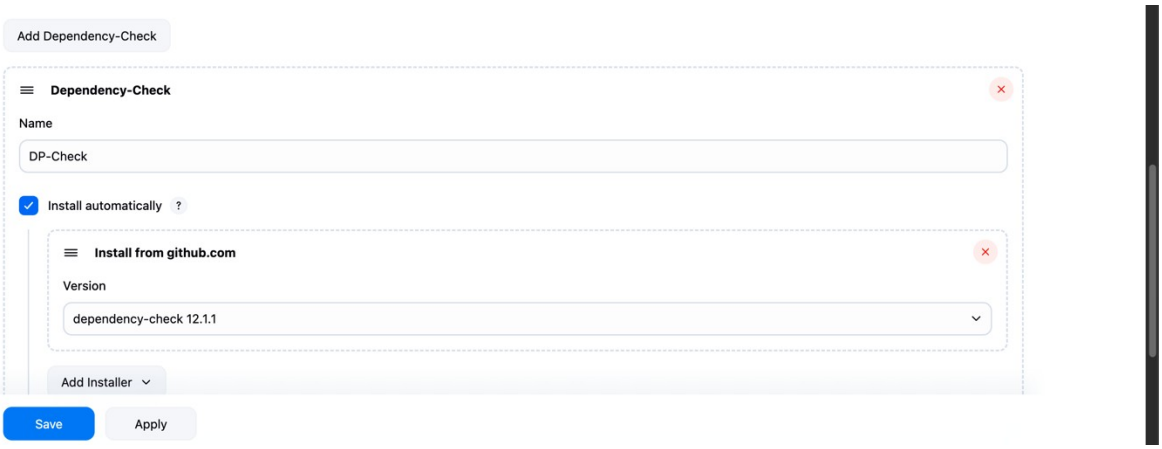
Go to manage jenkins and click on tools to add the Dependency check tool from github



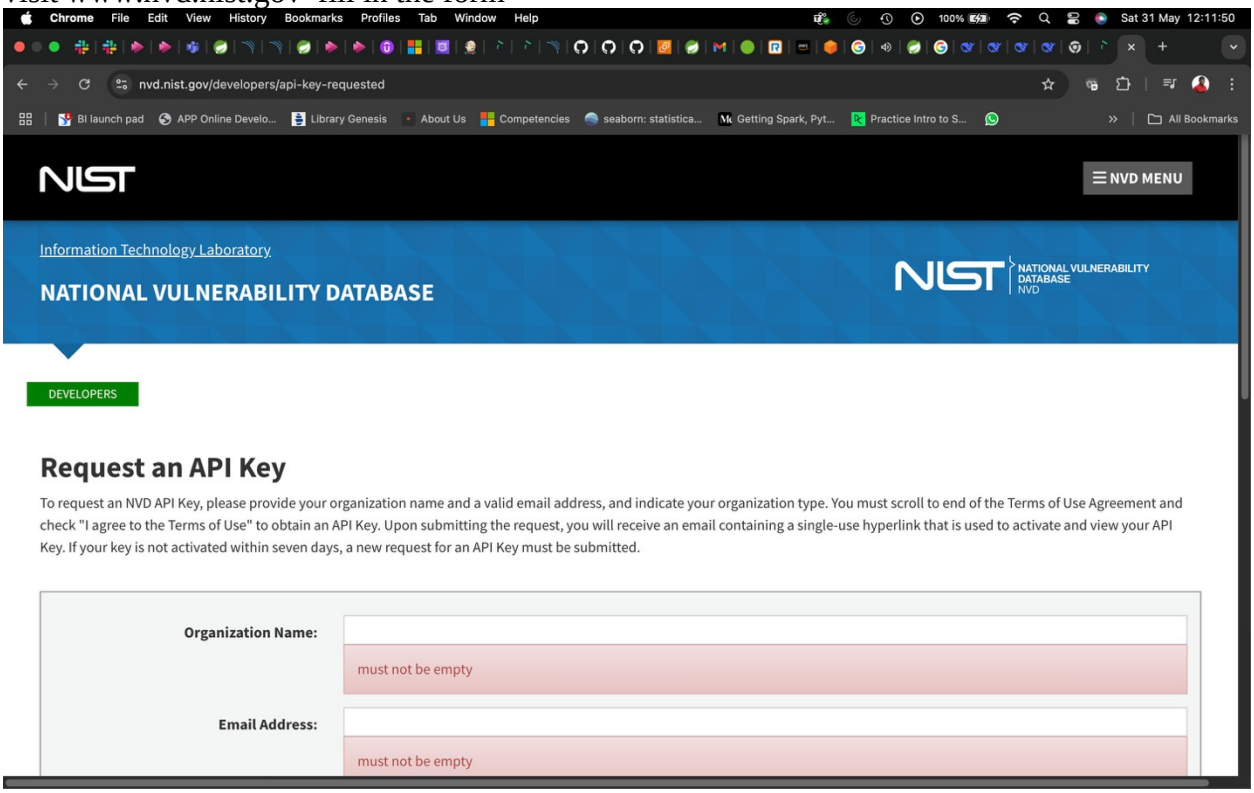
give the dependency check a name, click on "install automatically" and install from github



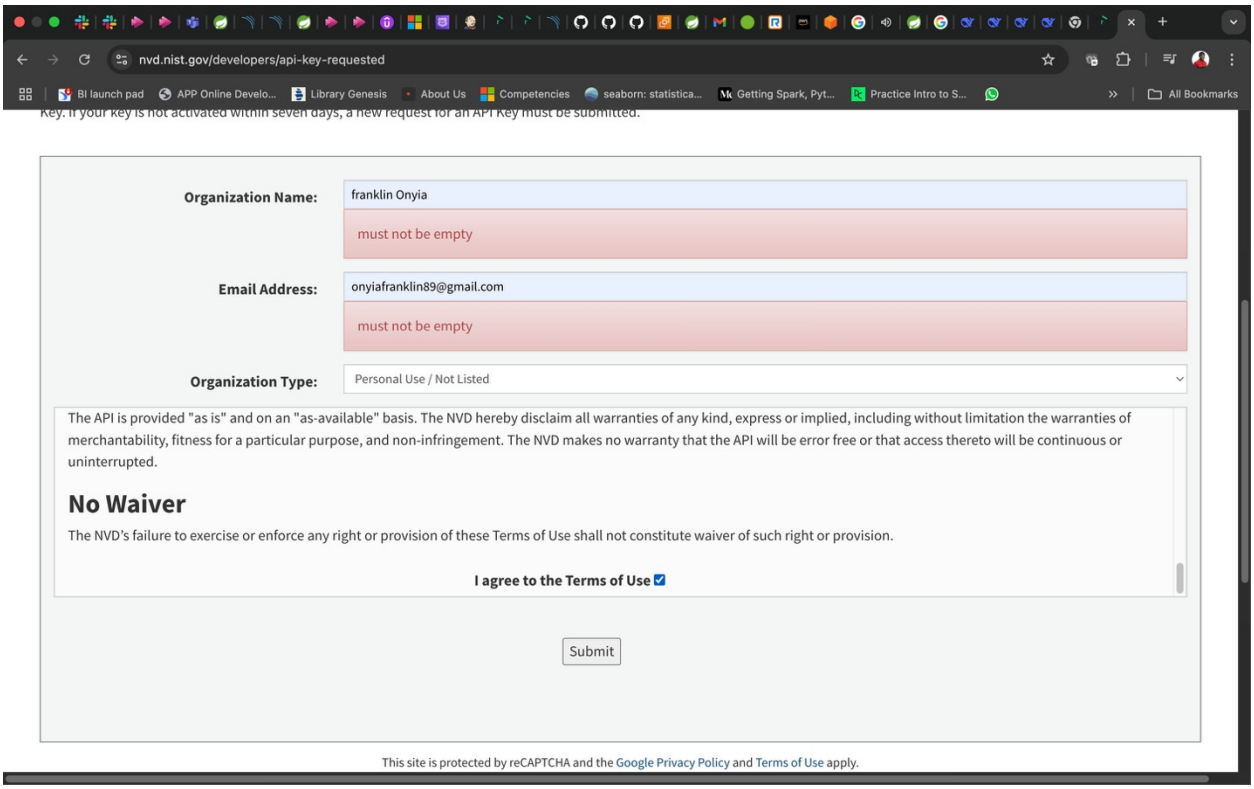
Install with latest or stable version



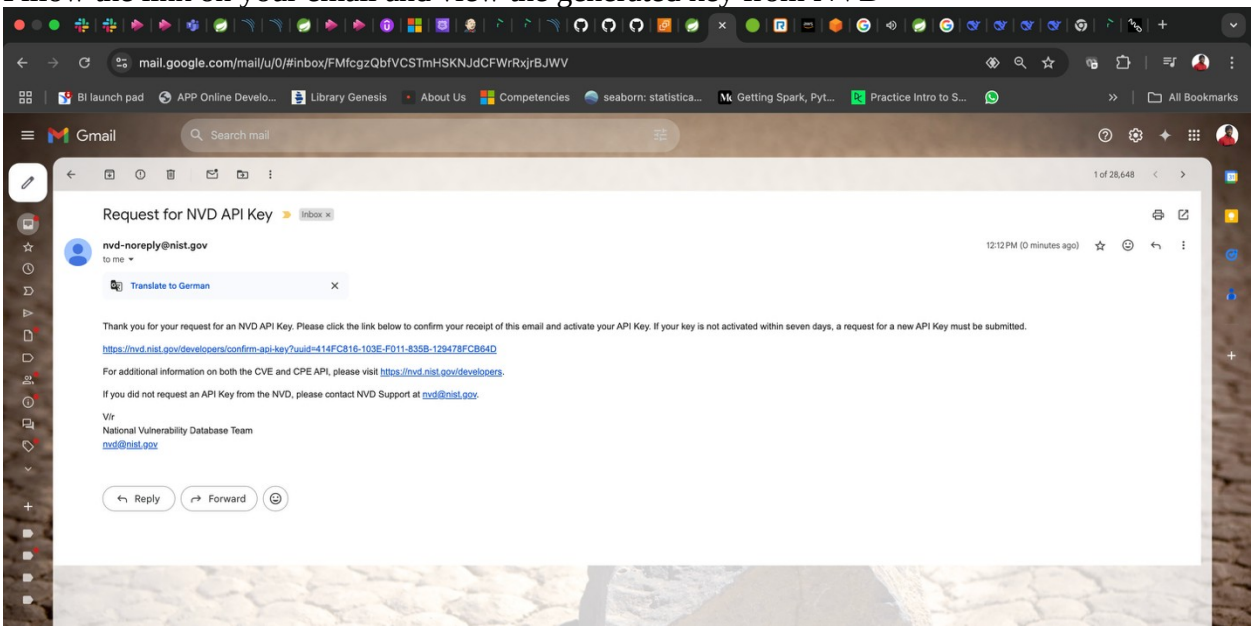
Next is to get the NVD key that the DP check scan would use for the scanning.
visit www.nvd.nist.gov fill in the form



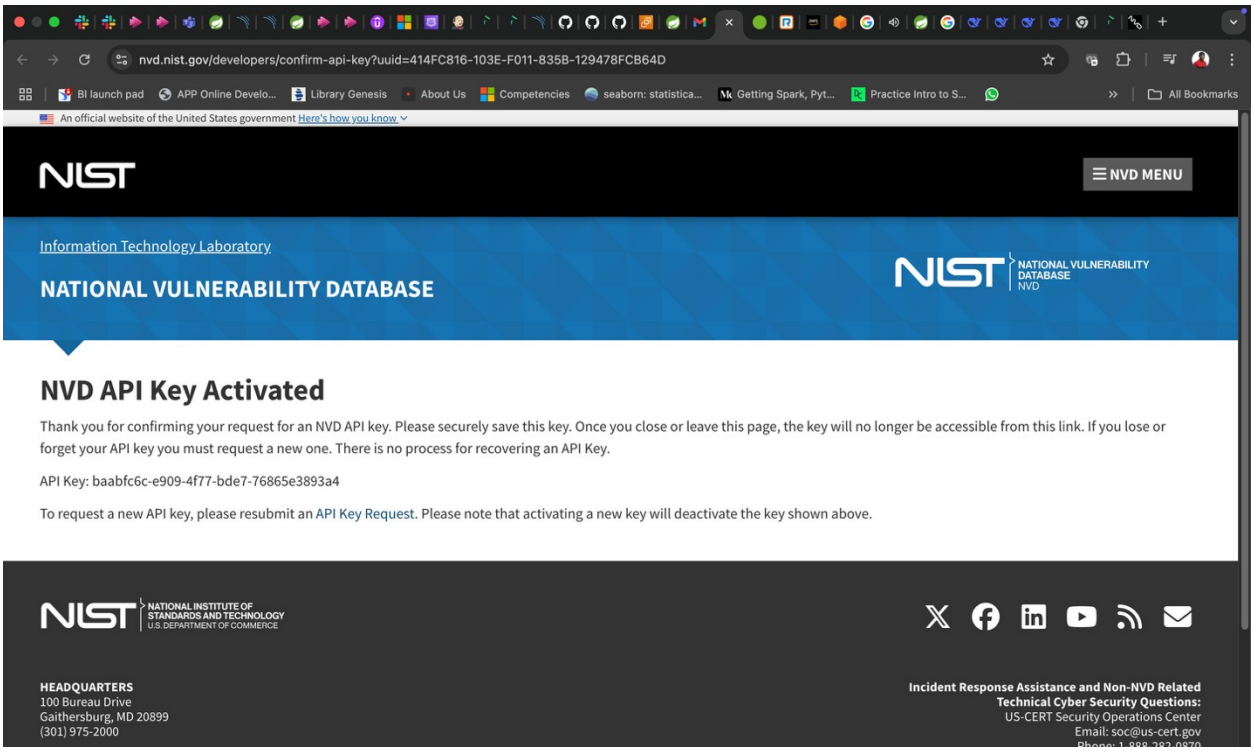
Agree to the terms and condition and submit. You would get a notification via the email you registered it with.



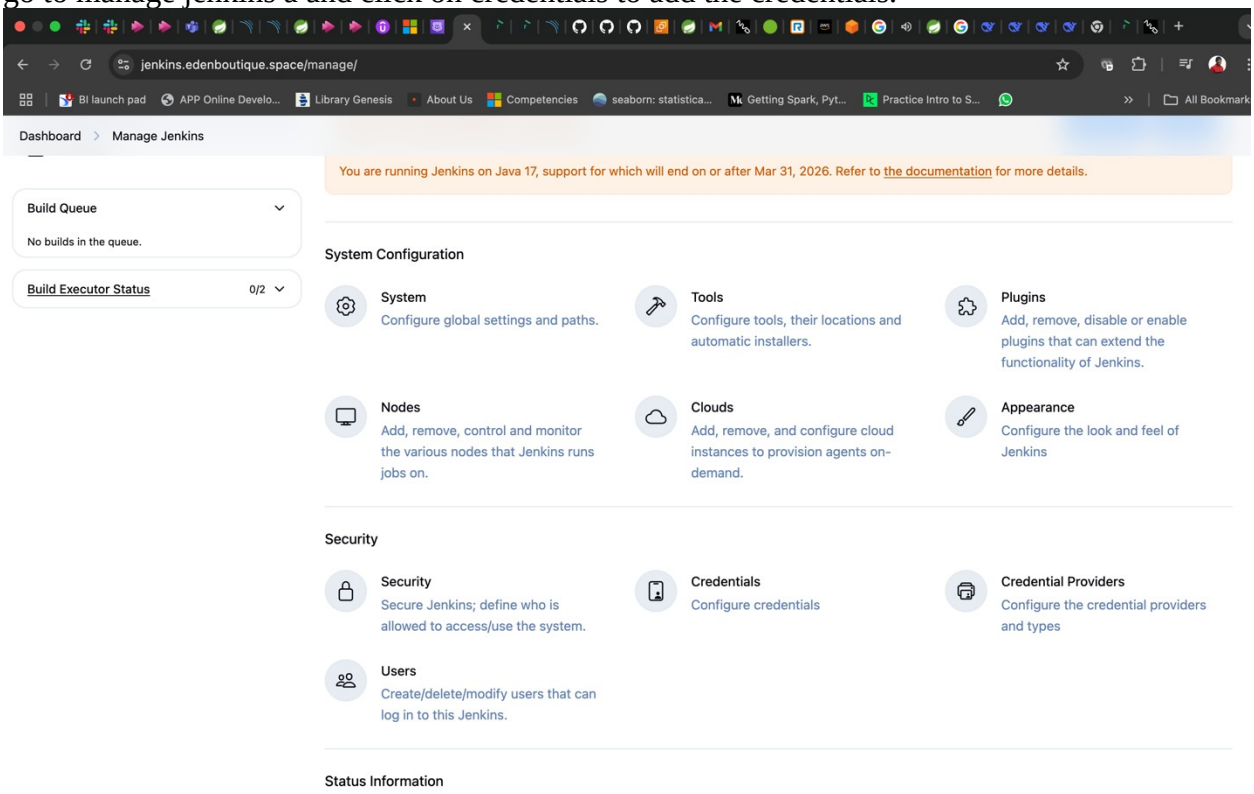
Fillow the link on your email and view the generated key from NVD



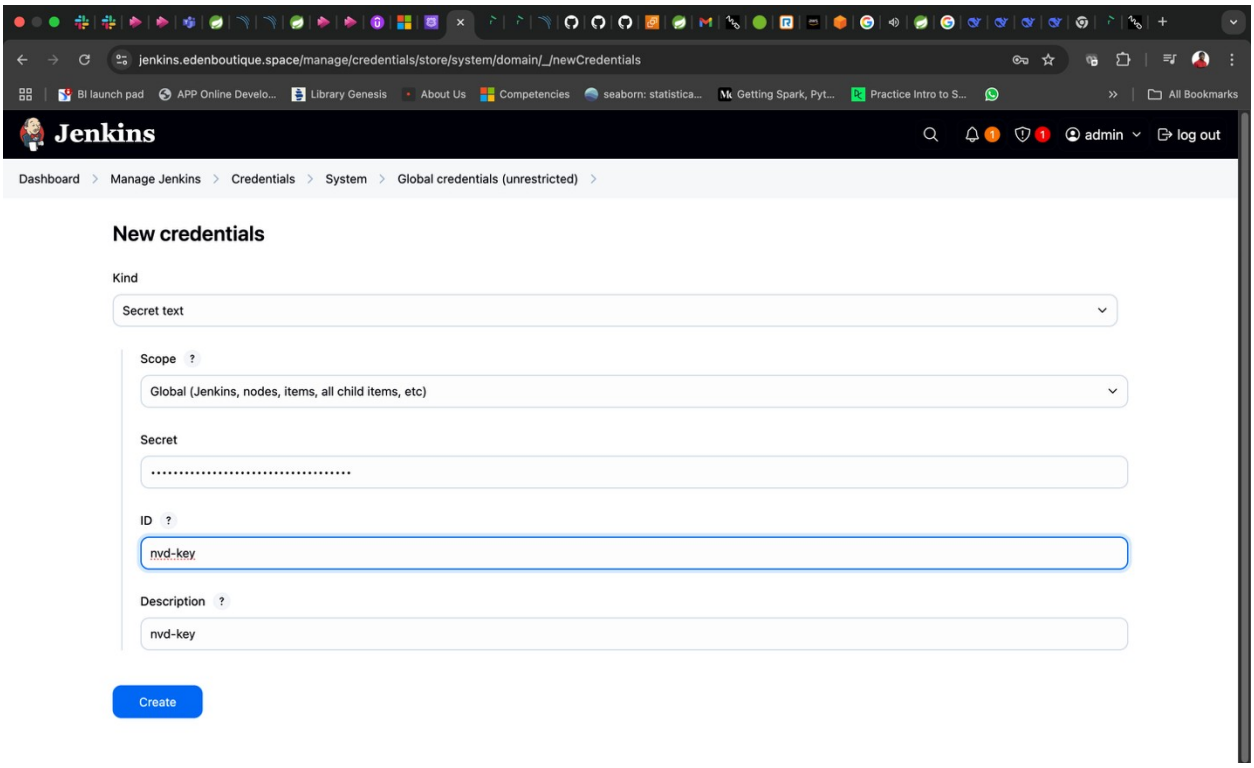
Copy the key



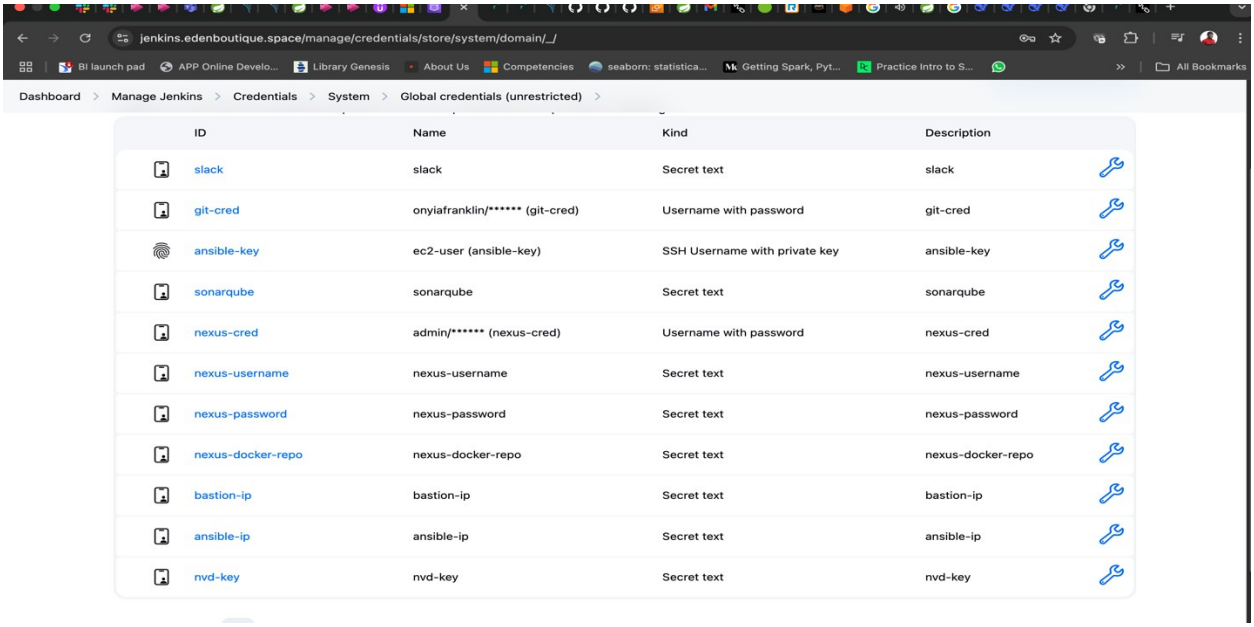
Copy the ket and use it to create a secret text credentials for the nvd key .
go to manage jenkins a and click on credentials to add the credentials.



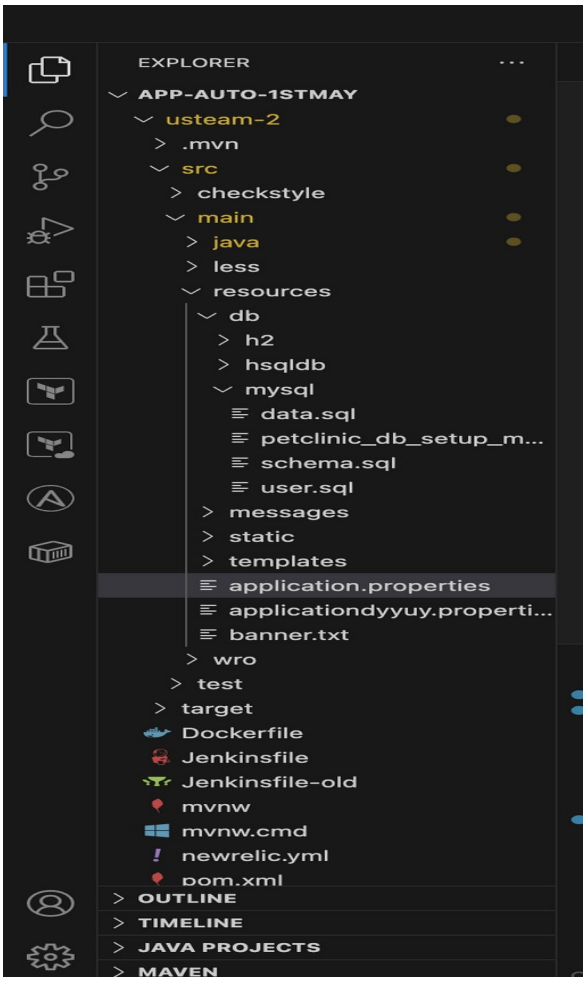
Give the credental a name and paste the key



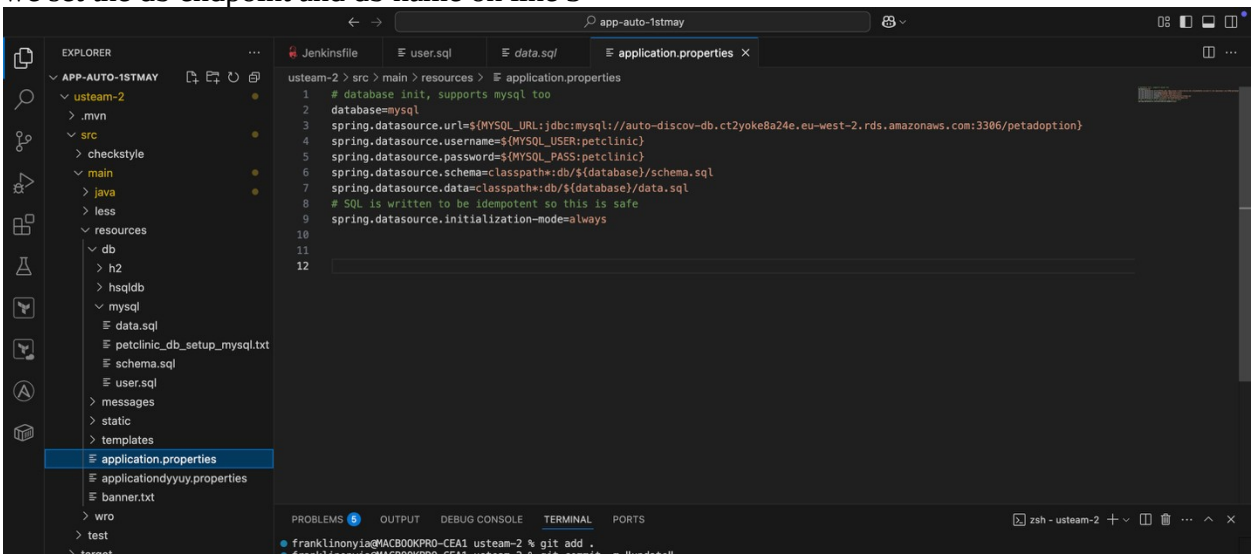
View all credentials created



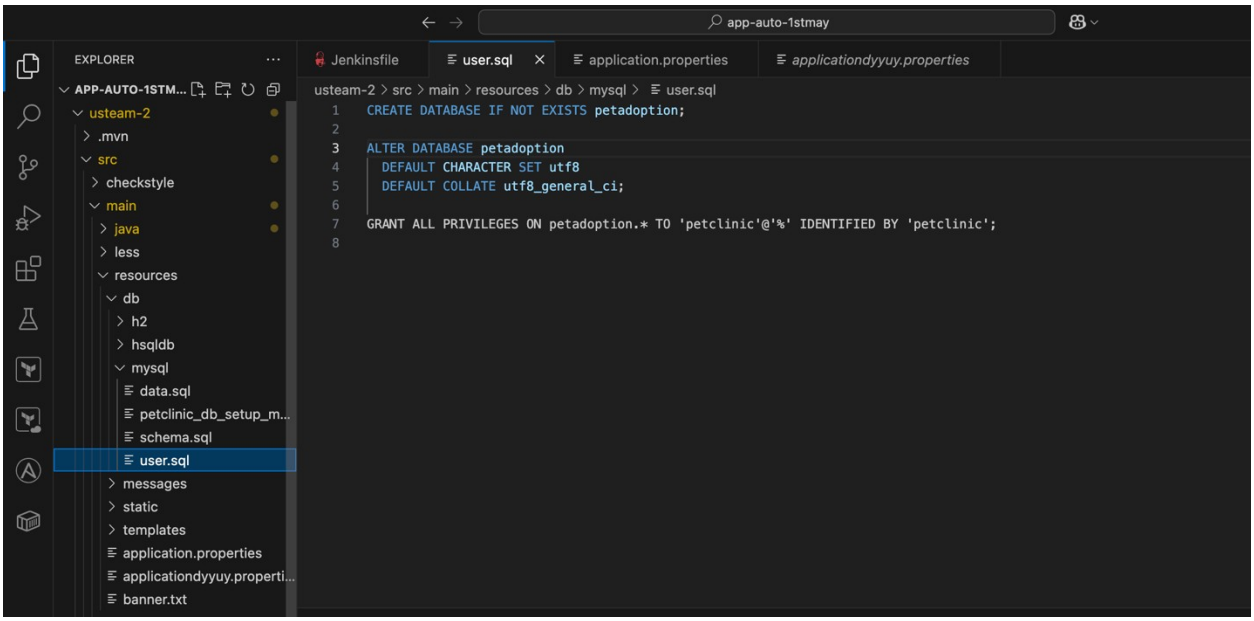
Next is to go to the application code on vscode to set the mysql for your application to persist data on the created database.
On the application code, go to the “application.properties” to set the database url endpoint and database name.



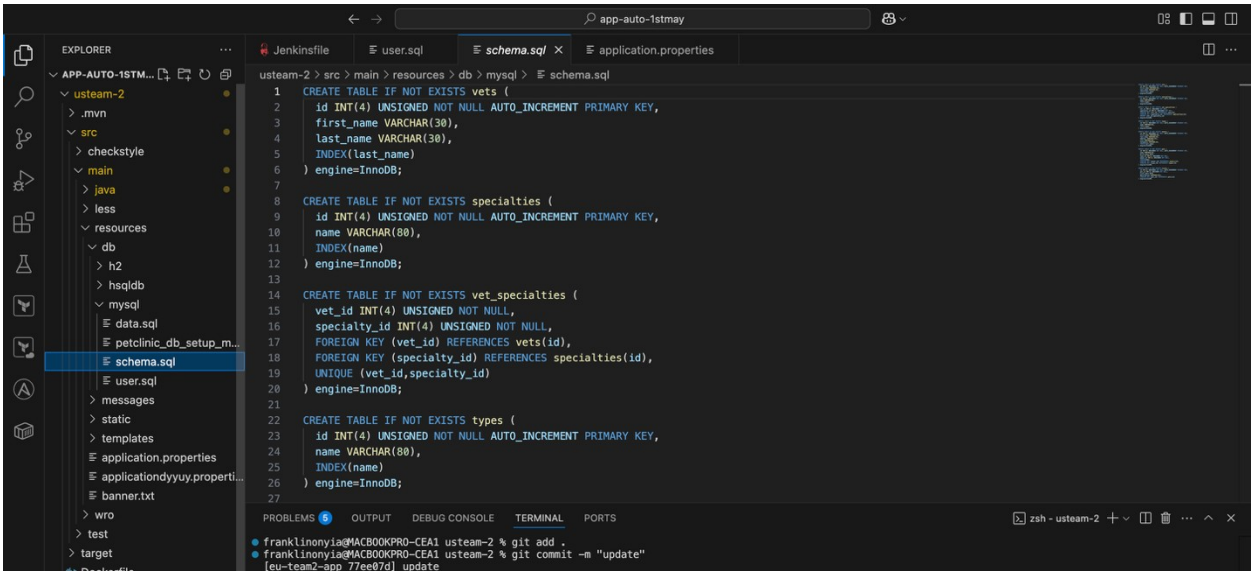
we set the db endpoint and db name on line 3



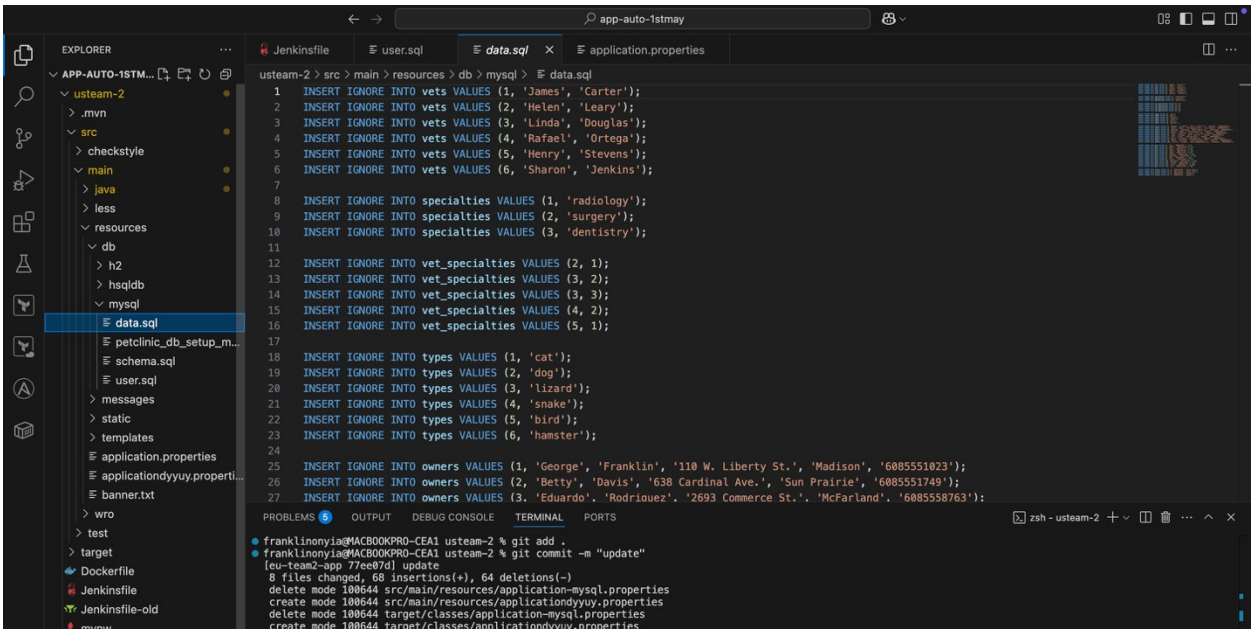
On the mysql, make sure you the database name exist on the user.sql



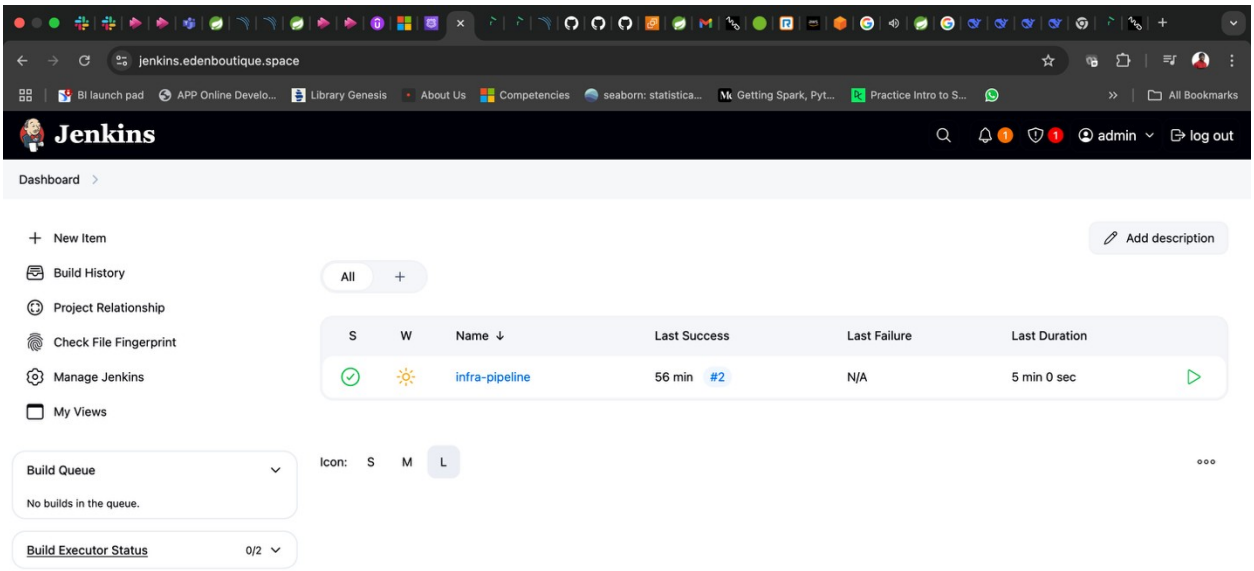
also check the database schema



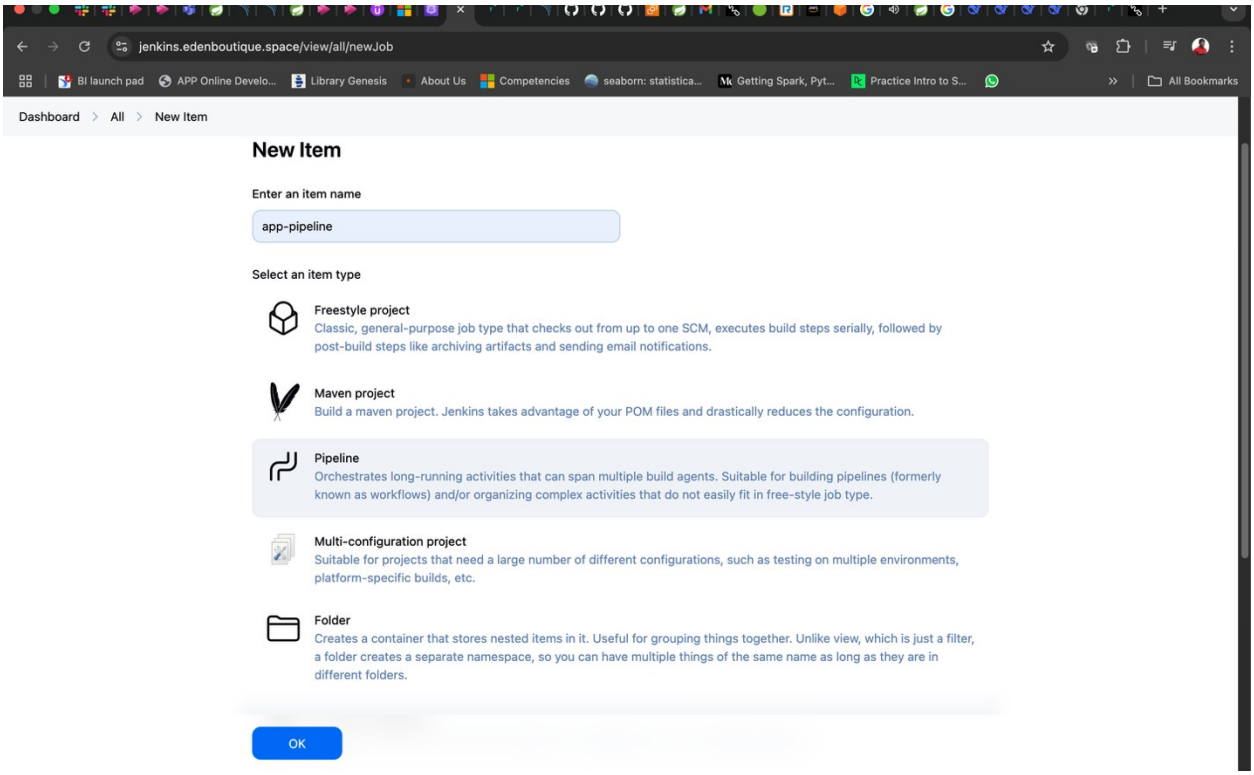
also view the previous data on data.sql



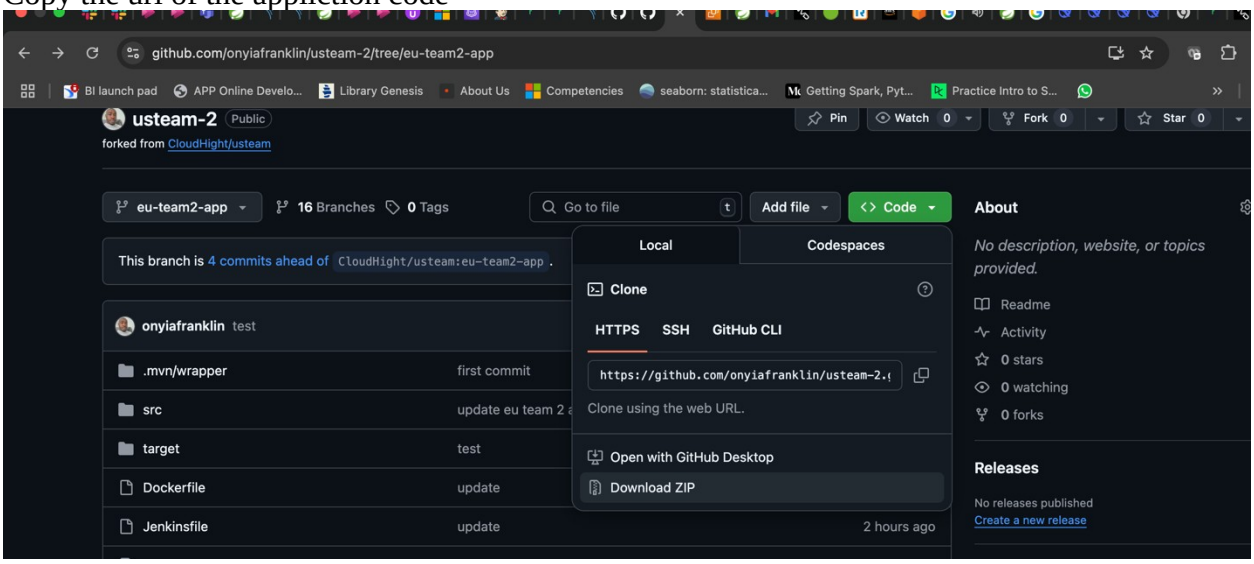
Now is time to create the application pipeline
click on “new item”



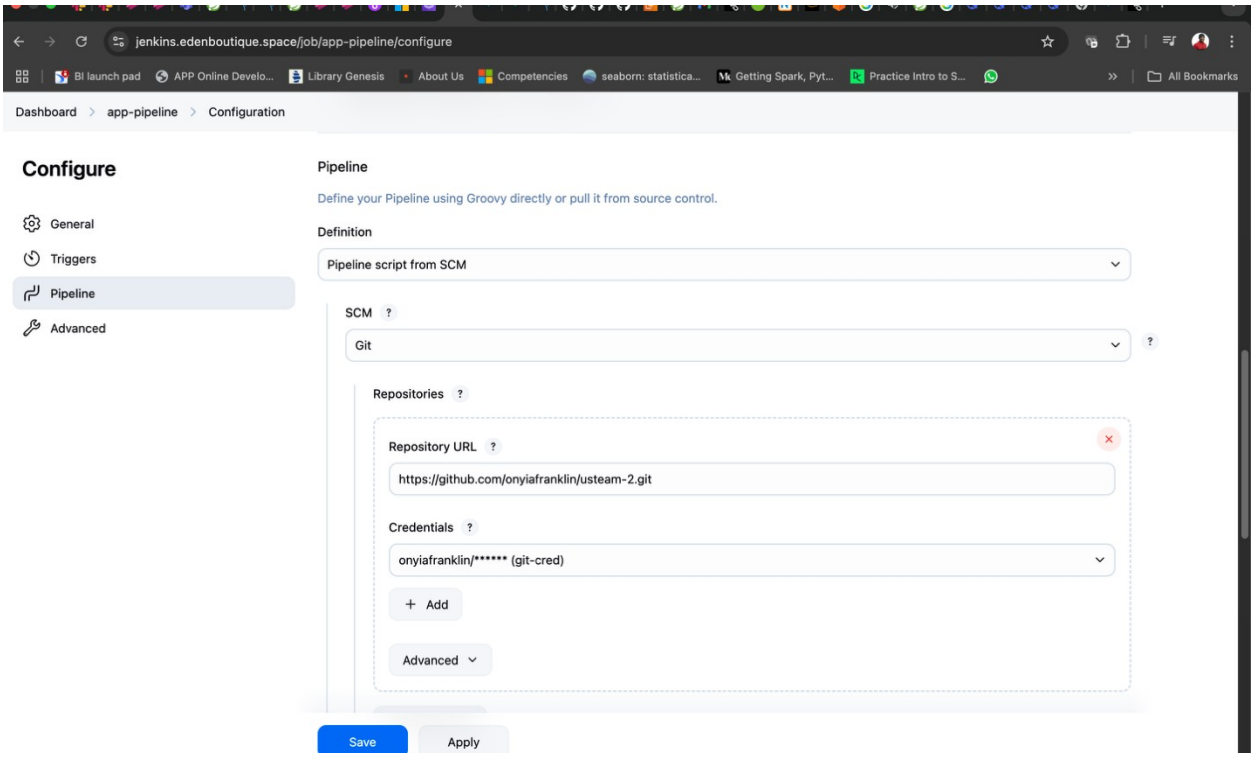
Give your new pipeline a name



Copy the url of the appliction code

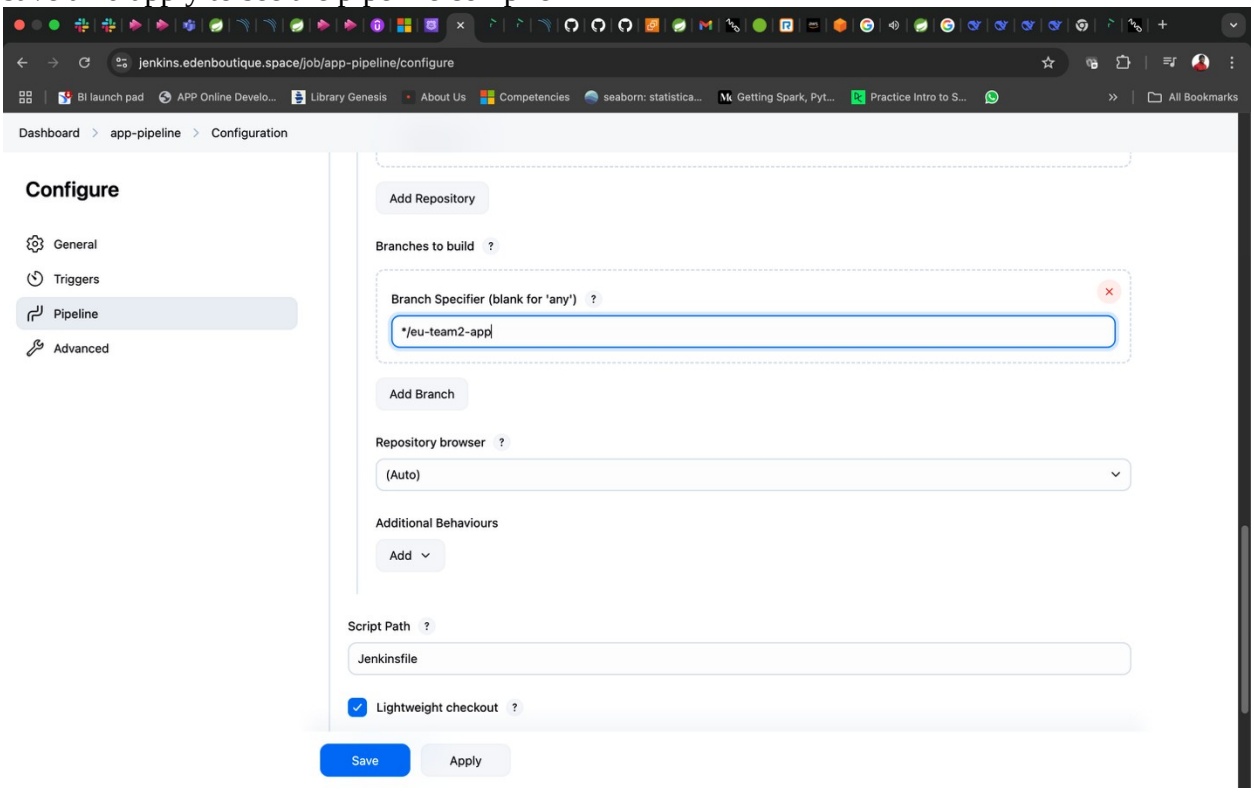


add the url to the repository url and add your git credentials



add the branch you would like to build and add the script path you would like to build which is Jenkinsfile

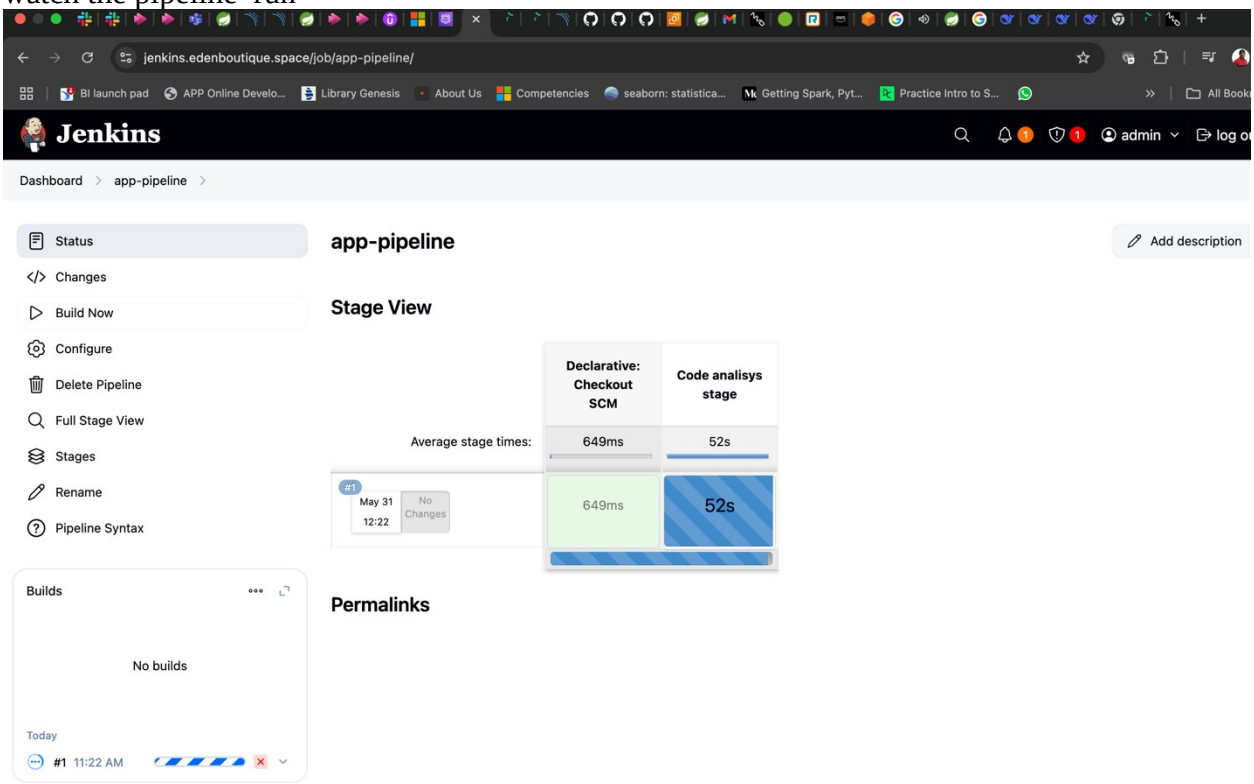
save a nd apply to see the pipeline compile



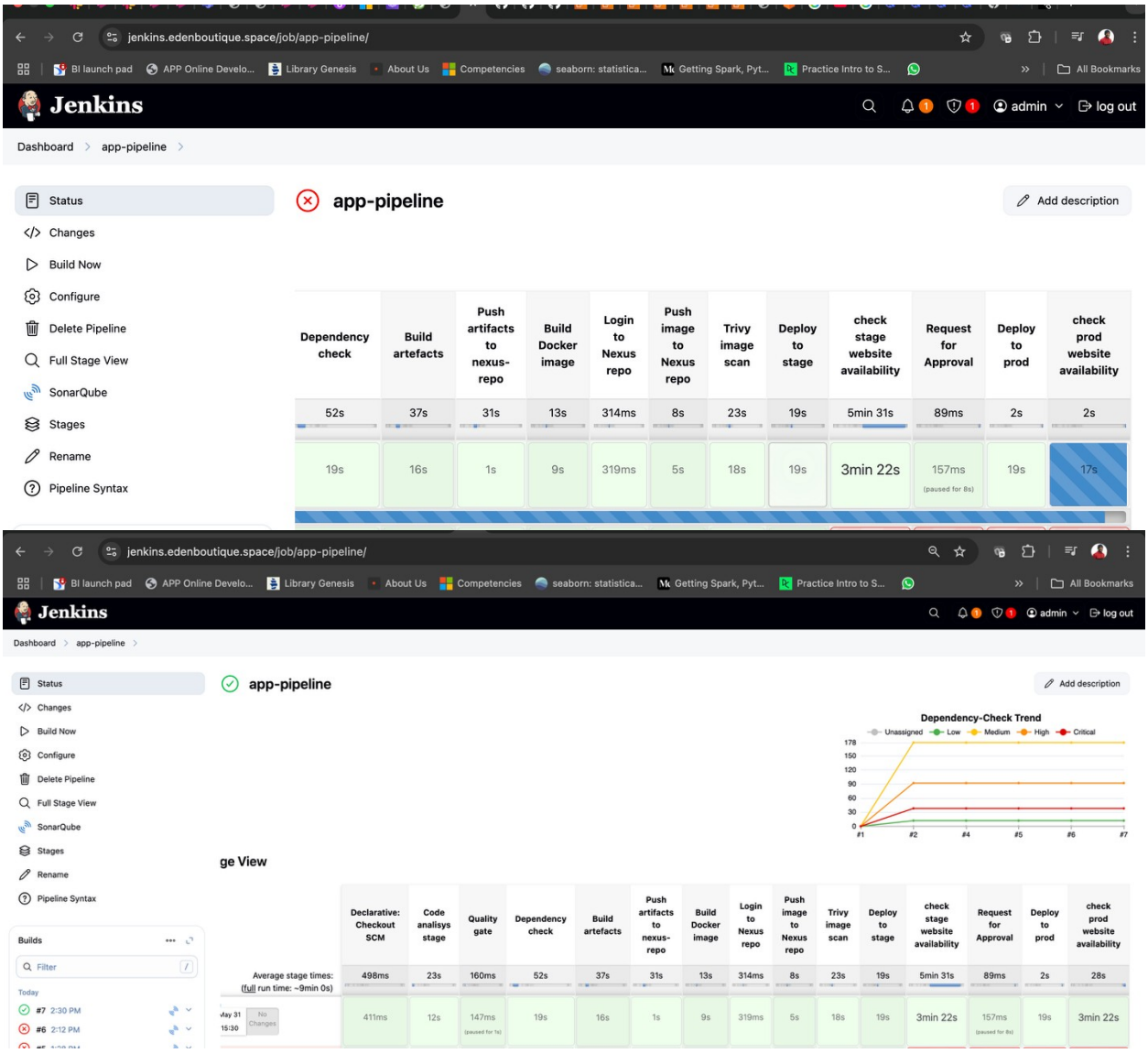
```
management capabilities on your system.
View your connected systems at https://console.redhat.com/insights

You can learn more about how to register your system
using rhc at https://red.ht/registration
[ec2-user@Nexus ~]$ sudo cat /app/sonatype-work/nexus3/admin.password
0a300d52-1efa-41ce-af7d-74ca3d65638d[ec2-user@Nexus ~]$ exit
logout
Connection to 10.0.1.156 closed.
[ec2-user@bastion ~]$ exit
logout
Connection to 18.130.51.153 closed.
[ec2-user@jenkins infra-pipeline]$ sudo systemctl restart docker
[ec2-user@jenkins infra-pipeline]$
```

0 ⚠ 0 Ln 1, Col 27 (26 selected) Spaces: 4

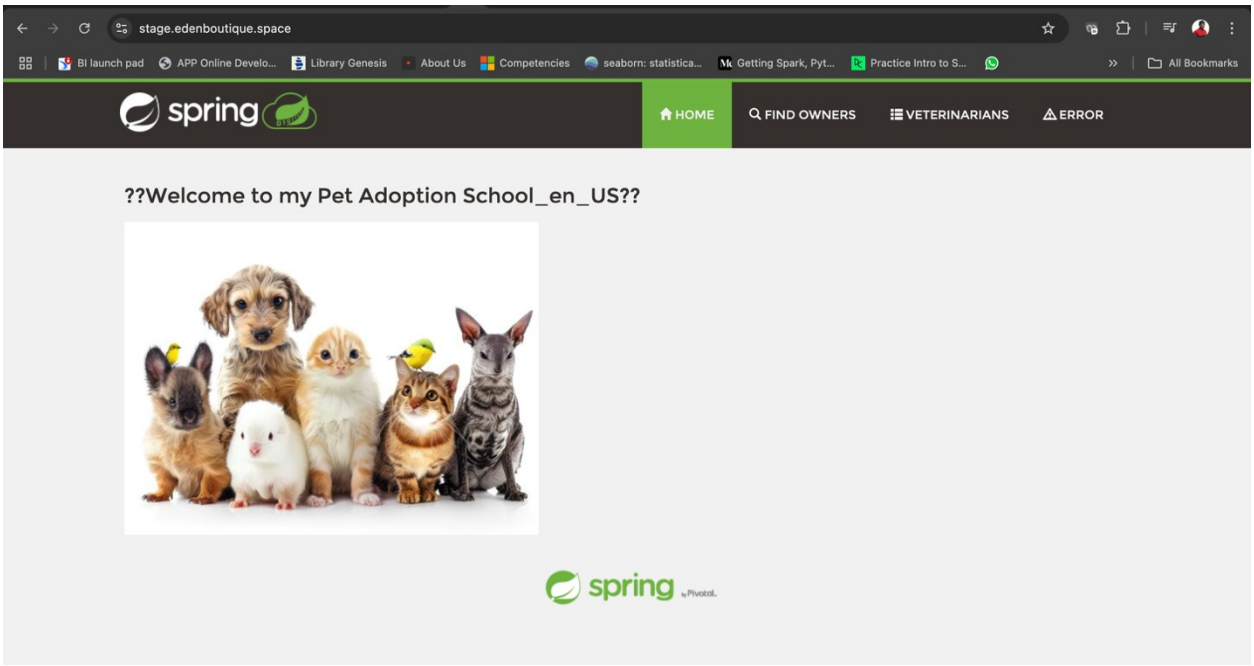
watch the pipeline run



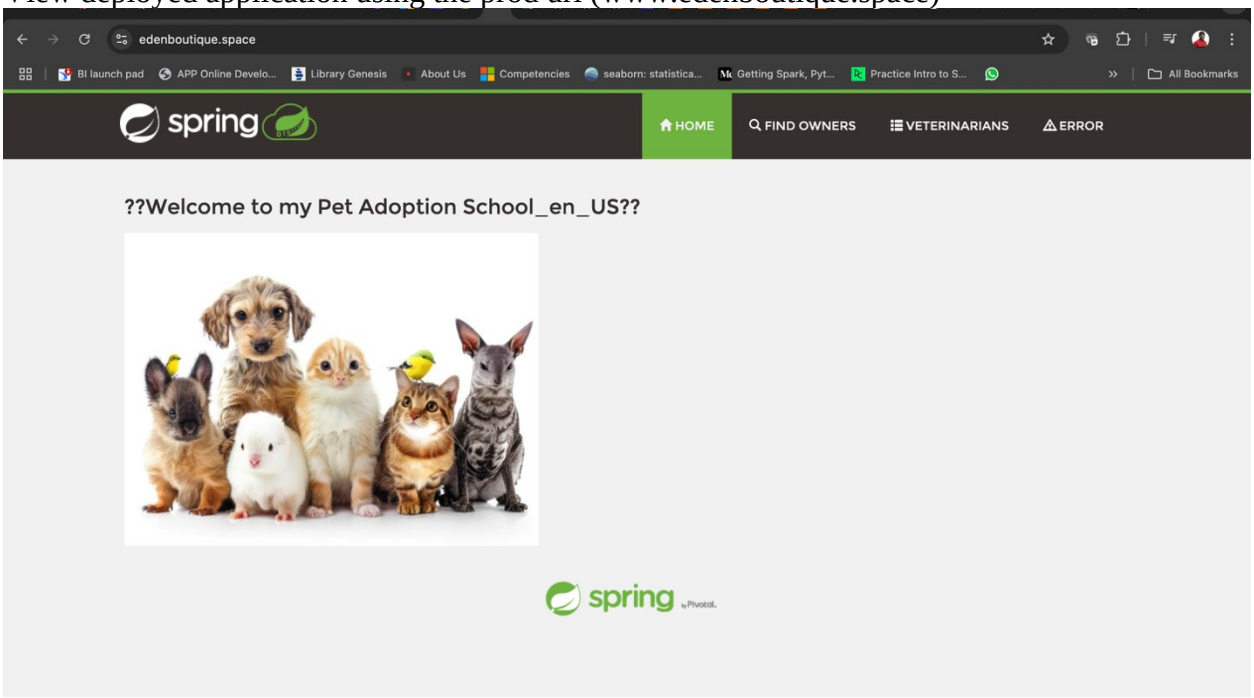
before deploy to production, it should request for approval.
thisnis important because ches such as user acceptability test, load test performance, security and compliance testing,as well as smoke test is done before approval to production environment



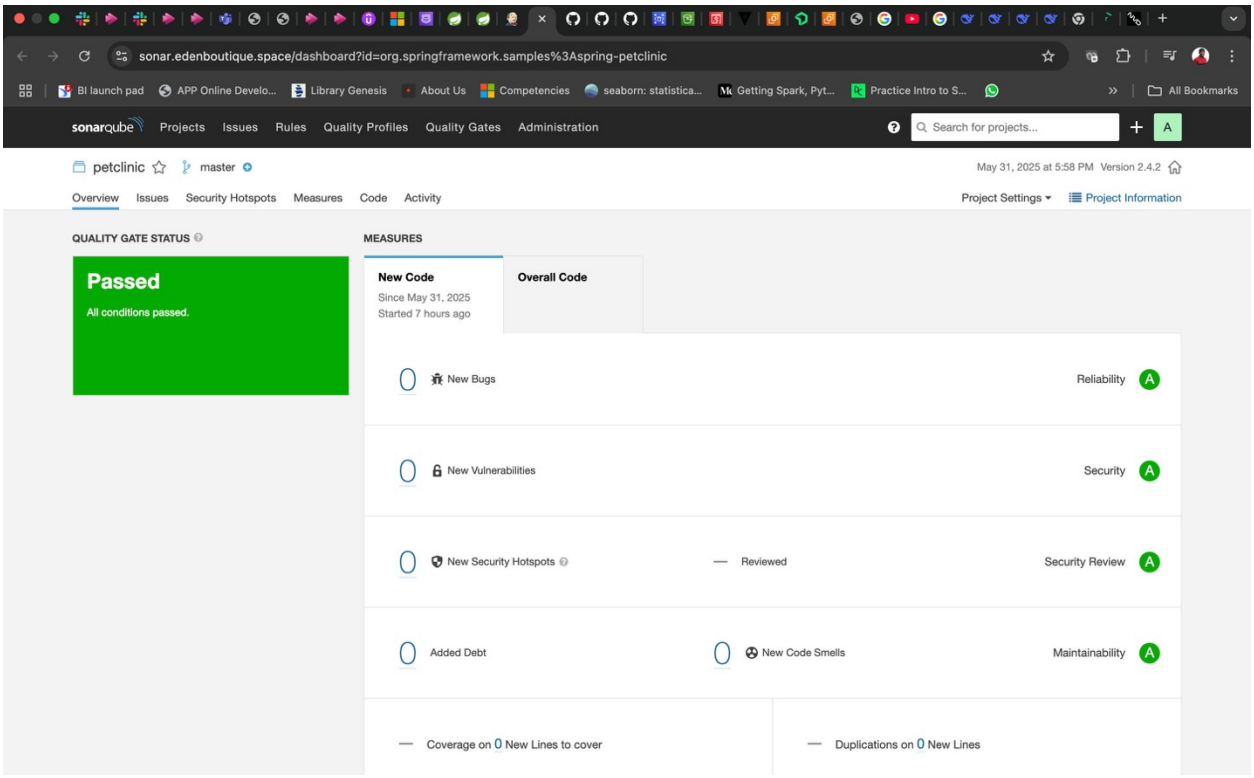
View deployed application using the stage url (stage.edenboutique.space)



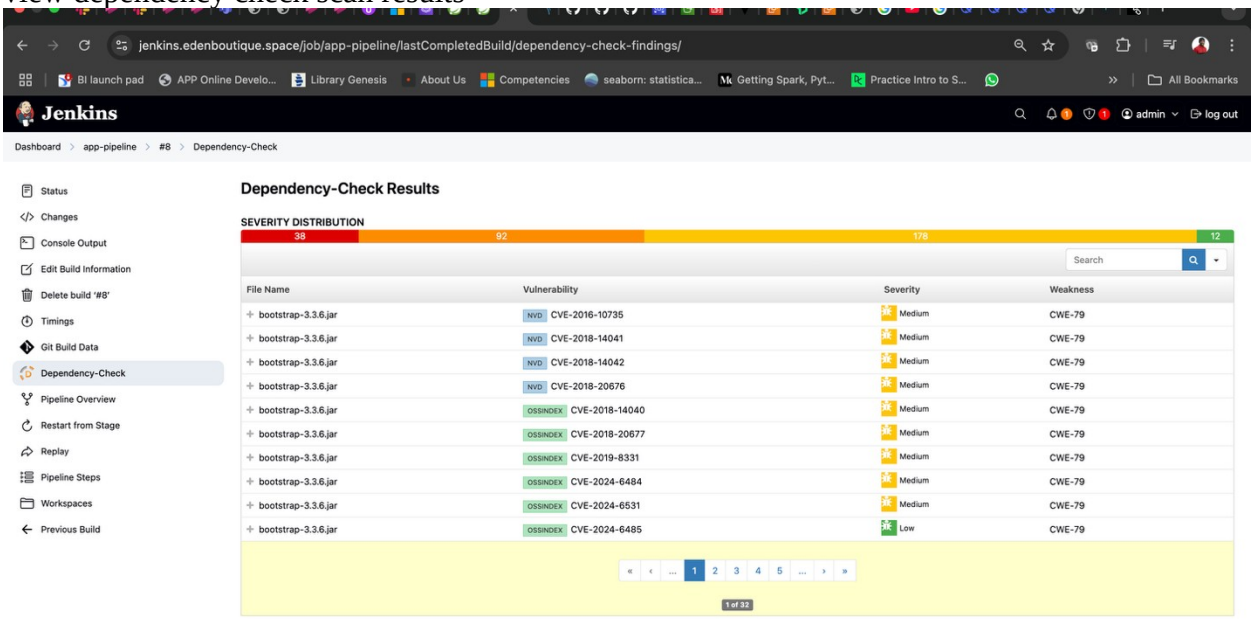
View deployed application using the prod url (www.edenboutique.space)



view quality gate scan results



view dependency check scan results



K. PERSISTING OUR DATA TO OUR DATABASE

Now lets check if our database persisted data

login to the stage server through the bastion host

```
-rw-r--r--.  1 root jenkins  300 May 31 10:18 variable.tf
drwxr-xr-x.  2 root jenkins  127 May 31 10:14 vault-jenkins
[ec2-user@jenkins infra-pipeline]$ ssh -i auto-discov-key.pem ec2-user@18.130.51.153
Register this system with Red Hat Insights: insights-client --register
Create an account or view all your systems at https://red.ht/insights-dashboard
Last login: Sat May 31 10:48:53 2025 from 18.170.218.148
[ec2-user@bastion ~]$ ssh 10.0.4.220
The authenticity of host '10.0.4.220 (10.0.4.220)' can't be established.
ED25519 key fingerprint is SHA256:iKp473mQBUhVYobcpVqPw8AXhaCDUDgGv/yCUVG+wX0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.4.220' (ED25519) to the list of known hosts.
Register this system with Red Hat Insights: rhc connect

Example:
# rhc connect --activation-key <key> --organization <org>

The rhc client and Red Hat Insights will enable analytics and additional
management capabilities on your system.
View your connected systems at https://console.redhat.com/insights

You can learn more about how to register your system
using rhc at https://red.ht/registration
Last login: Sat May 31 17:01:29 2025
[ec2-user@stage-instance ~]$
```

onyiafranklin (1 week ago)

Install mysql

```
You can learn more about how to register your system
using rhc at https://red.ht/registration
Last login: Sat May 31 17:01:29 2025
[ec2-user@stage-instance ~]$ sudo yum install mysql -y
Updating Subscription Management repositories.
Unable to read consumer identity

This system is not registered with an entitlement server. You can use "rhc" or "subscription-manager" to register.

Last metadata expiration check: 1:59:16 ago on Sat 31 May 2025 03:13:01 PM UTC.
Dependencies resolved.
================================================================================
 Package                 Architecture          Version                 Repository
================================================================================
```

login to the database

```
Complete!
[ec2-user@stage-instance ~]$ mysql -h auto-discov-db.ct2yoke8a24e.eu-west-2.rds.amazonaws.com -u petclinic -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 149
Server version: 5.7.44-rds.20250213 MySQL 5.7 is now under RDS Extended Support. Please upgrade to 8.0 to benefit from the latest RDS innovation

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

show dtabase

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 149
Server version: 5.7.44-rds.20250213 MySQL 5.7 is now under RDS Extended Support. Please upgrade to 8.0 to benefit from the latest RDS innov

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| innodb             |
| mysql              |
| performance_schema |
| petadoption        |
| sys                |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

onyiafranklin (1 week ago) Ln 91, Col 12 Spaces: 4 UTF-8

show tables

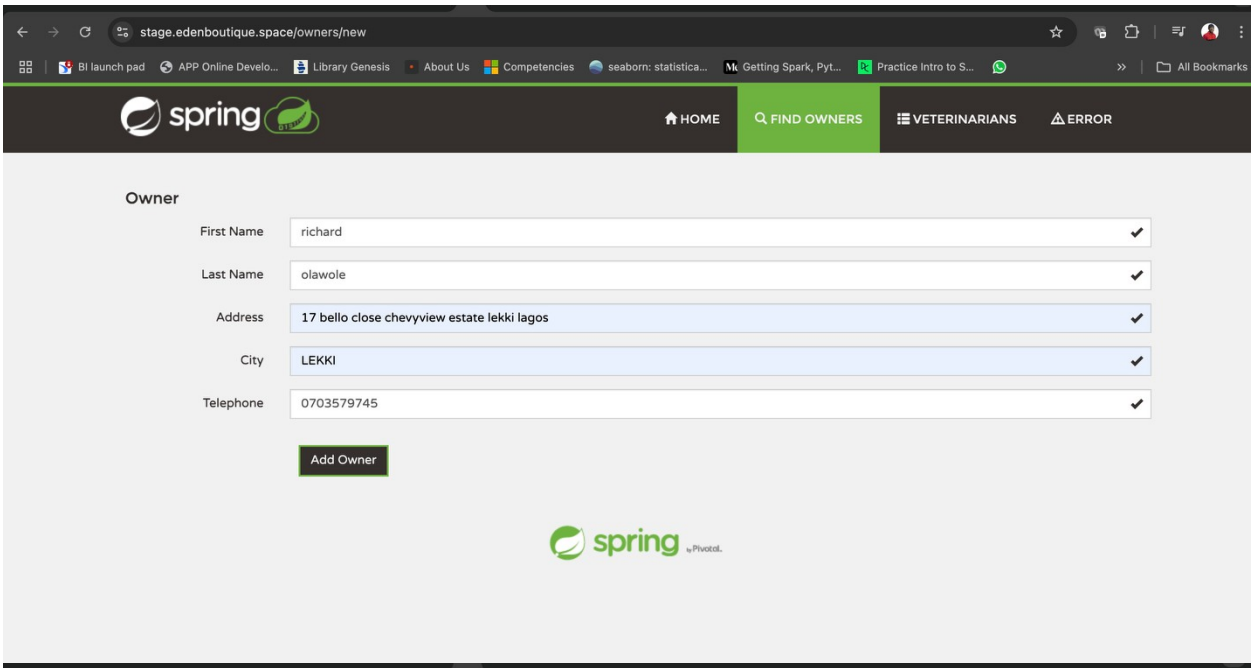
```
6 rows in set (0.00 sec)

mysql> use petadoption
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------------+
| Tables_in_petadoption |
+------------------------+
| owners                 |
| pets                   |
| specialties            |
| types                  |
| vet_specialties        |
| vets                   |
| visits                 |
+------------------------+
7 rows in set (0.00 sec)

mysql>
```

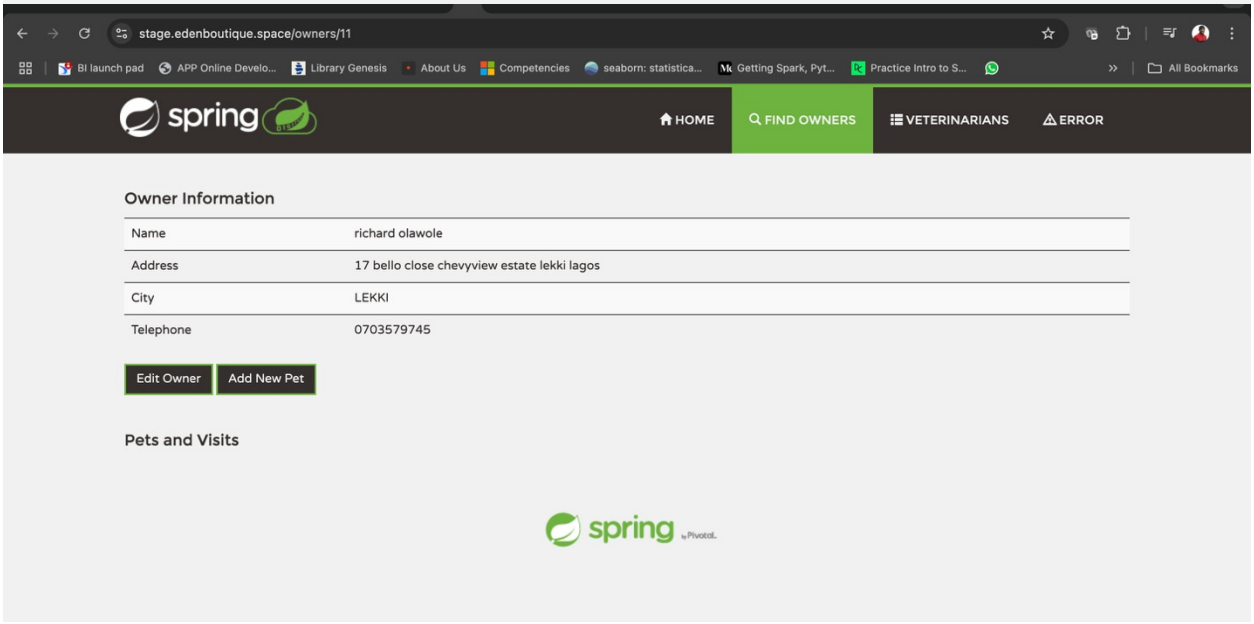
onyiafranklin (1 wee

select to see owners table

```
| vets                   |
| visits                 |
+------------------------+
7 rows in set (0.00 sec)

mysql> select * from owners
    -> ;
+----+------------+-----------+----------------------+-------------+------------+
| id | first_name | last_name | address              | city        | telephone  |
+----+------------+-----------+----------------------+-------------+------------+
|  1 | George     | Franklin  | 110 W. Liberty St.   | Madison     | 6085551023 |
|  2 | Betty      | Davis     | 638 Cardinal Ave.    | Sun Prairie | 6085551749 |
|  3 | Eduardo    | Rodriquez | 2693 Commerce St.    | McFarland   | 6085558763 |
|  4 | Harold     | Davis     | 563 Friendly St.     | Windsor     | 6085553198 |
|  5 | Peter      | McTavish  | 2387 S. Fair Way     | Madison     | 6085552765 |
|  6 | Jean       | Coleman   | 105 N. Lake St.      | Monona      | 6085552654 |
|  7 | Jeff       | Black     | 1450 Oak Blvd.       | Monona      | 6085555387 |
|  8 | Maria      | Escobito  | 345 Maple St.        | Madison     | 6085557683 |
|  9 | David      | Schroeder | 2749 Blackhawk Trail | Madison     | 6085559435 |
| 10 | Carlos     | Estaban   | 2335 Independence La. | Waunakee    | 6085555487 |
+----+------------+-----------+----------------------+-------------+------------+
10 rows in set (0.00 sec)

mysql>
```

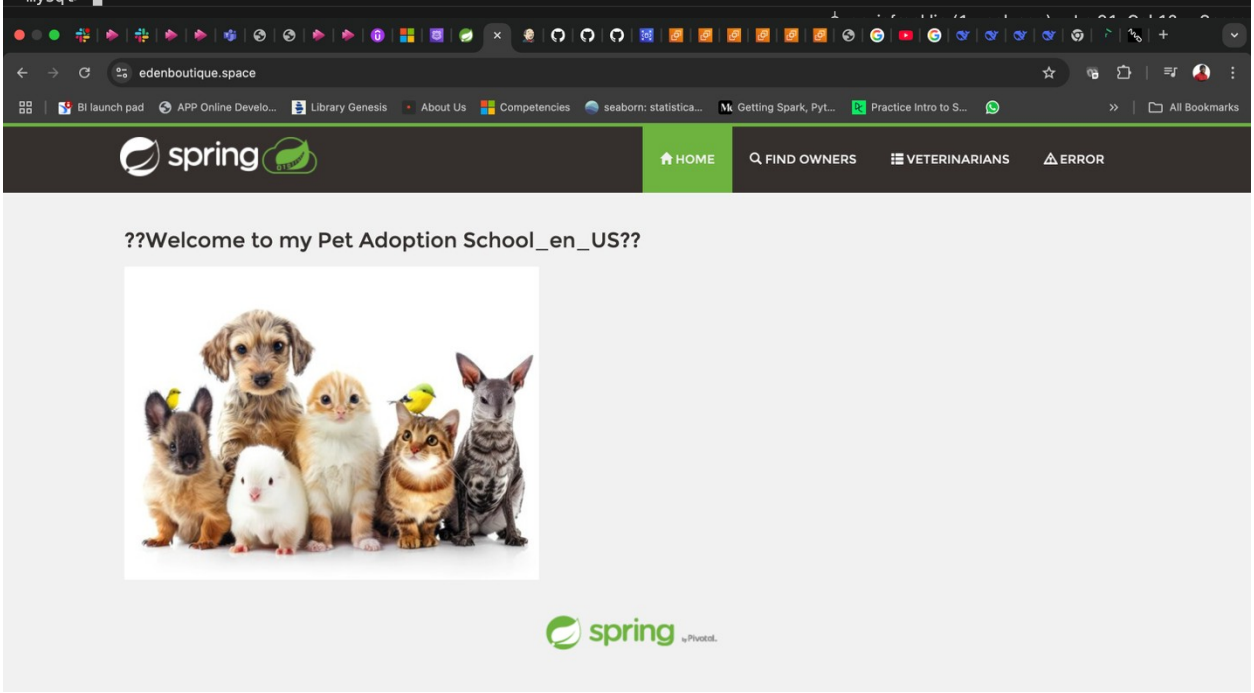
Now lets add data to the database and see if its persisted on the database
add data as shown below





select * from table to see Richard >> firstname and olawole>> last name added has been persisted

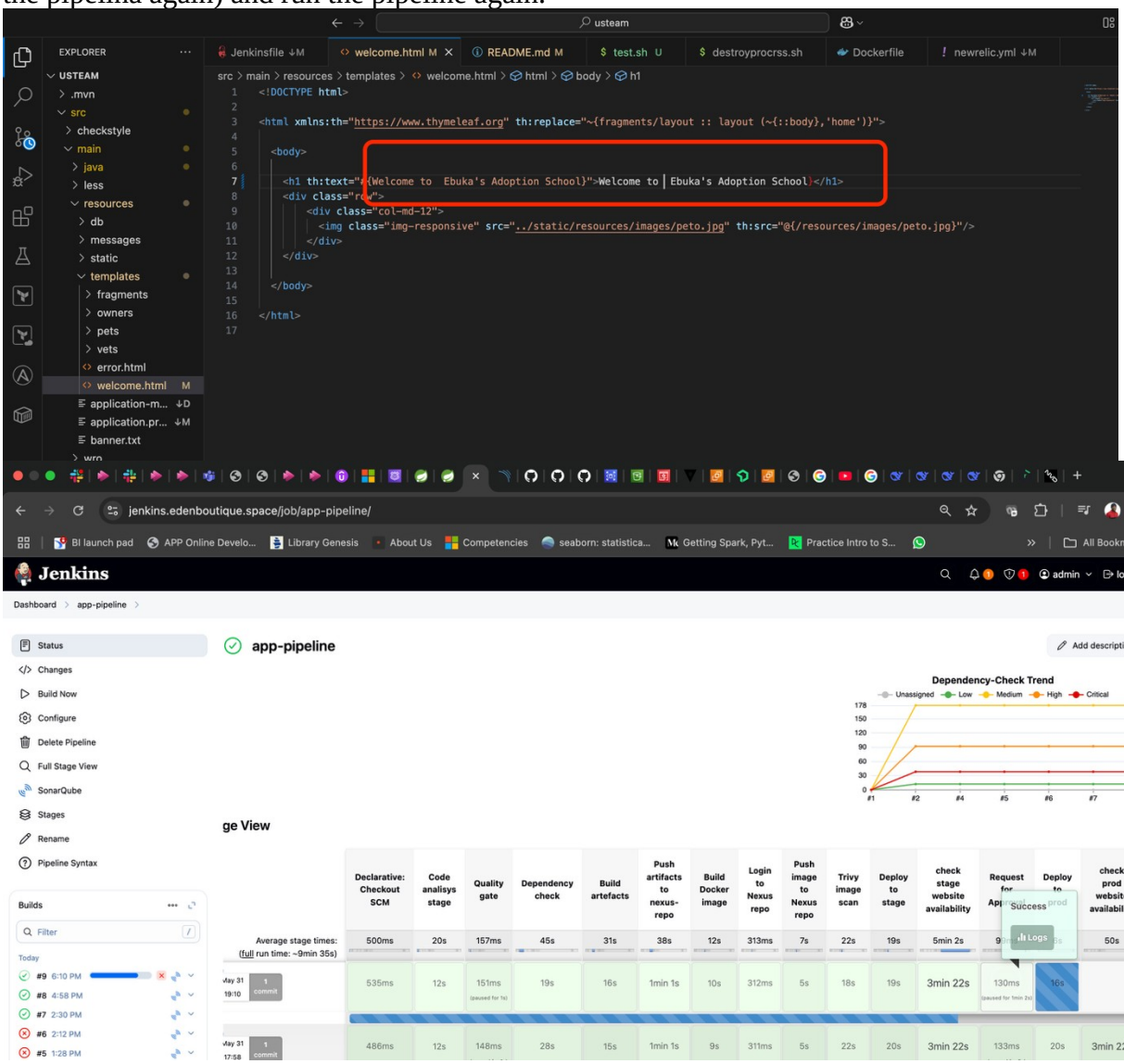
```
10 rows in set (0.00 sec)

mysql> select * from owners;
+----+------------+-----------+-----------------------------------------------+--------------+------------+
| id | first_name | last_name | address                                       | city         | telephone  |
+----+------------+-----------+-----------------------------------------------+--------------+------------+
|  1 | George     | Franklin  | 110 W. Liberty St.                            | Madison      | 6085551023 |
|  2 | Betty      | Davis     | 638 Cardinal Ave.                             | Sun Prairie  | 6085551749 |
|  3 | Eduardo    | Rodriquez | 2693 Commerce St.                             | McFarland    | 6085558763 |
|  4 | Harold     | Davis     | 563 Friendly St.                              | Windsor      | 6085553198 |
|  5 | Peter      | McTavish  | 2387 S. Fair Way                              | Madison      | 6085552765 |
|  6 | Jean       | Coleman   | 105 N. Lake St.                               | Monona       | 6085552654 |
|  7 | Jeff       | Black     | 1450 Oak Blvd.                                | Monona       | 6085555387 |
|  8 | Maria      | Escobito  | 345 Maple St.                                 | Madison      | 6085557683 |
|  9 | David      | Schroeder | 2749 Blackhawk Trail                          | Madison      | 6085559435 |
| 10 | Carlos     | Estaban   | 2335 Independence La.                         | Waunakee     | 6085555487 |
| 11 | richard    | olawole   | 17 bello close chevyview estate lekki lagos   | LEKKI        | 0703579745 |
+----+------------+-----------+-----------------------------------------------+--------------+------------+
11 rows in set (0.00 sec)

mysql>
```



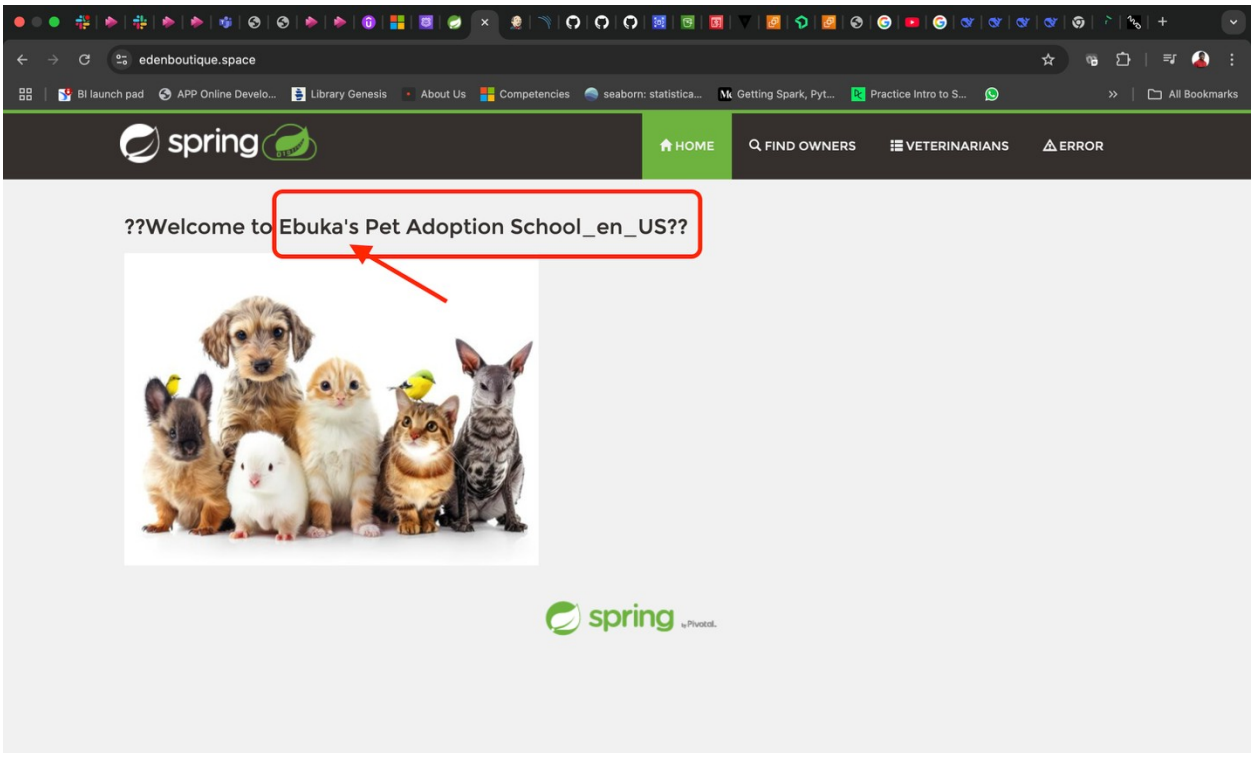
L. CHANGE ON APPLICATION PIPELINE AND REDEPLOYMENT TO VIEW CHANGE

Now lets change the welcome page tag name to (welcome to Ebuka’s adoption and run

the pipelina again) and run the pipeline again.



the change was effected



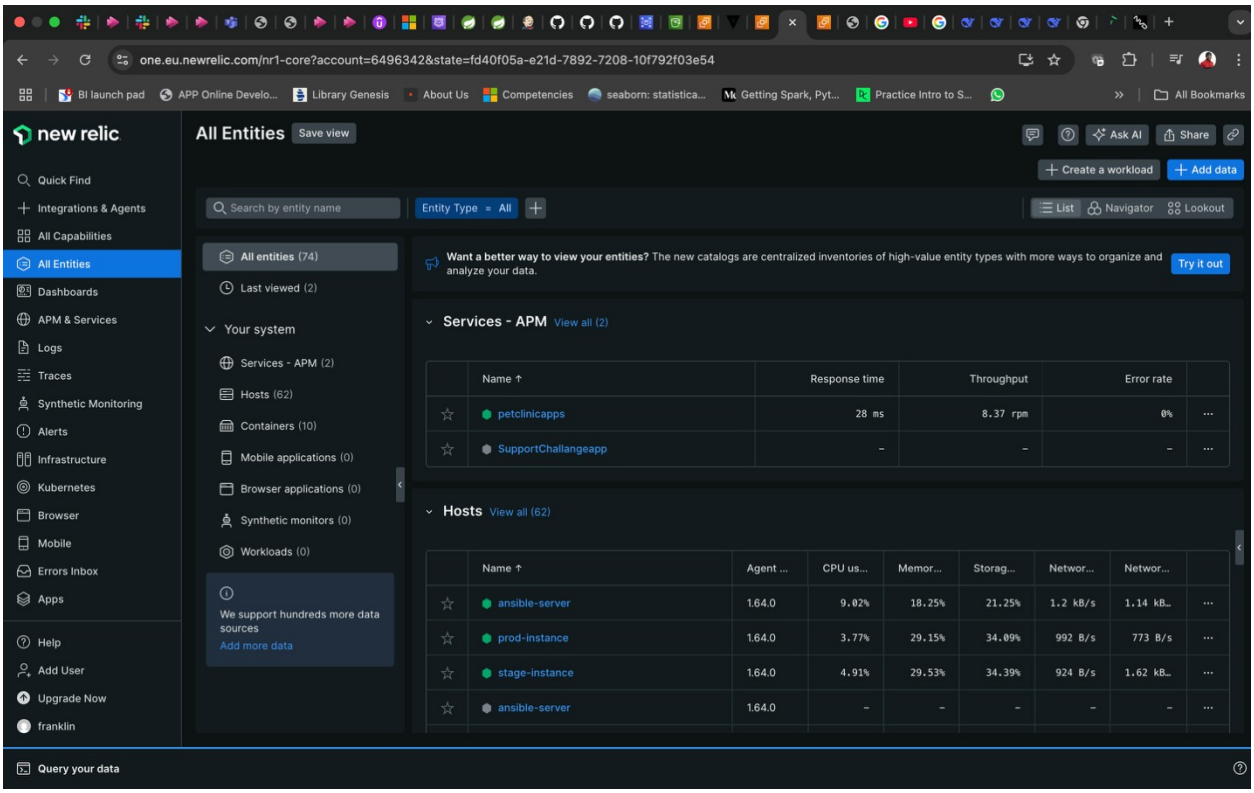
M. MONITORING WITH NEWRELIC

Application and infrastructure monitoring is very important to avoid disruption of service.

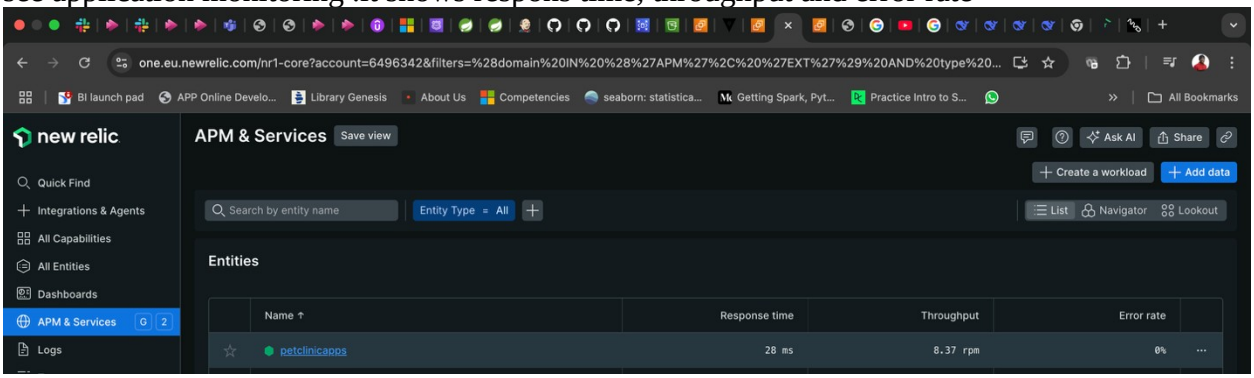
newrelic agent were installed on all server for monitoring and newrelic agent was also installed for application monitoring.

results of monitoring.

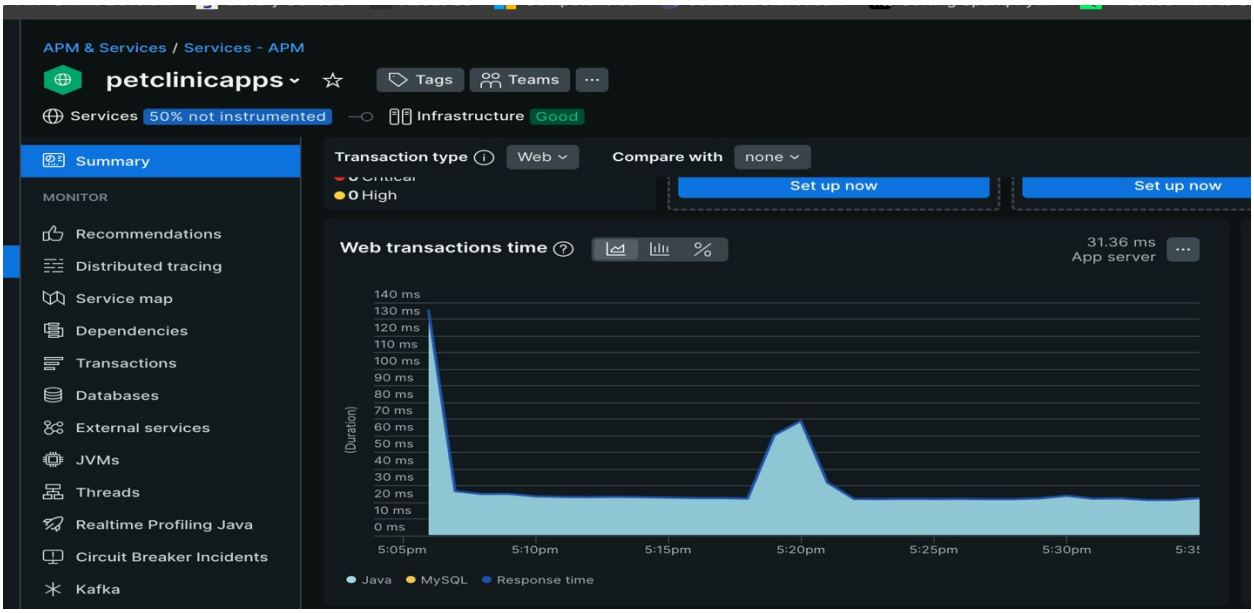
view all servers monitored showing the cpu usage, memory



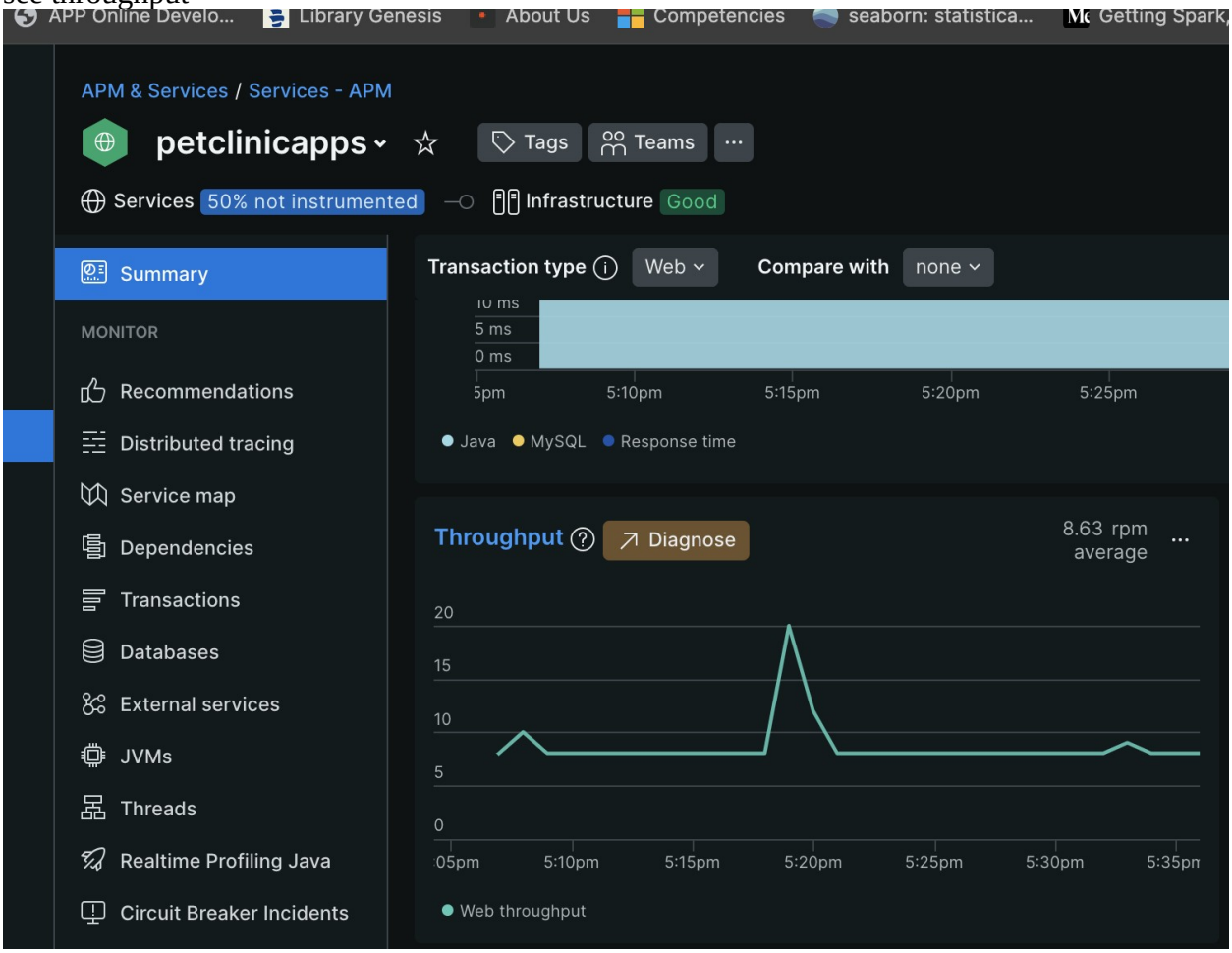
see application monitoring .it shows respons time, throughput and error rate



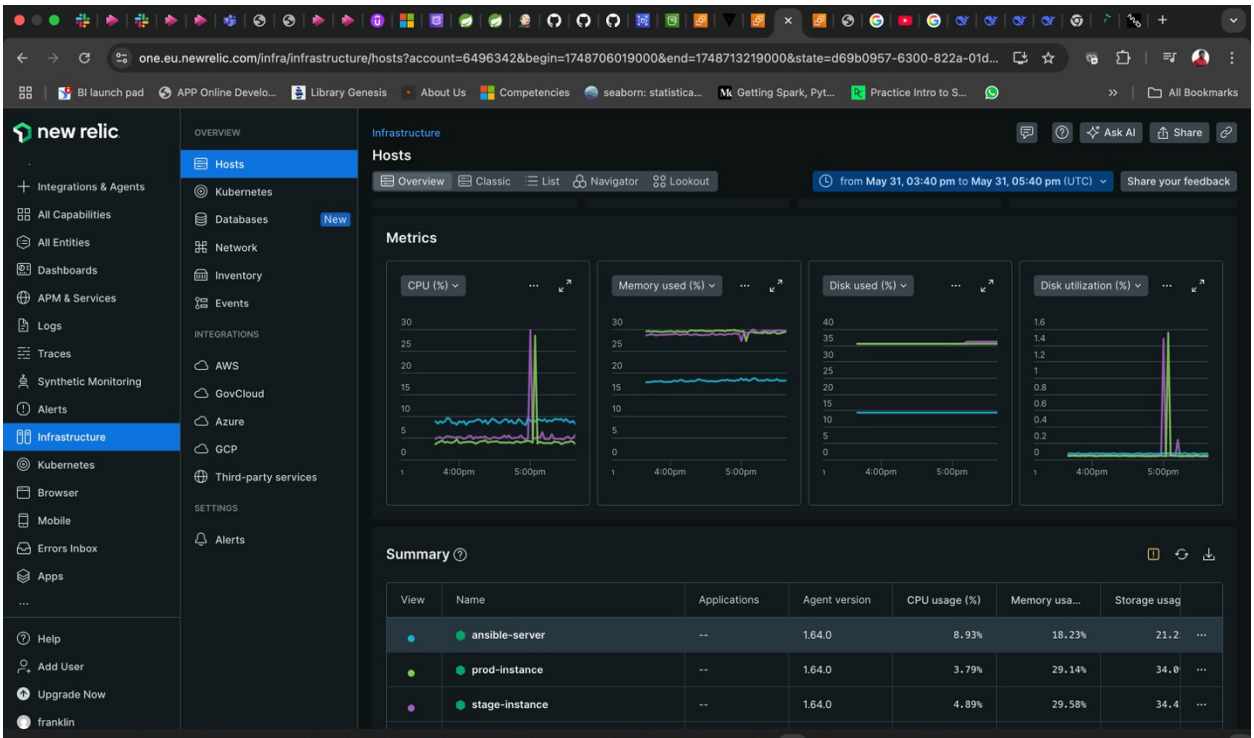
see web transaction time

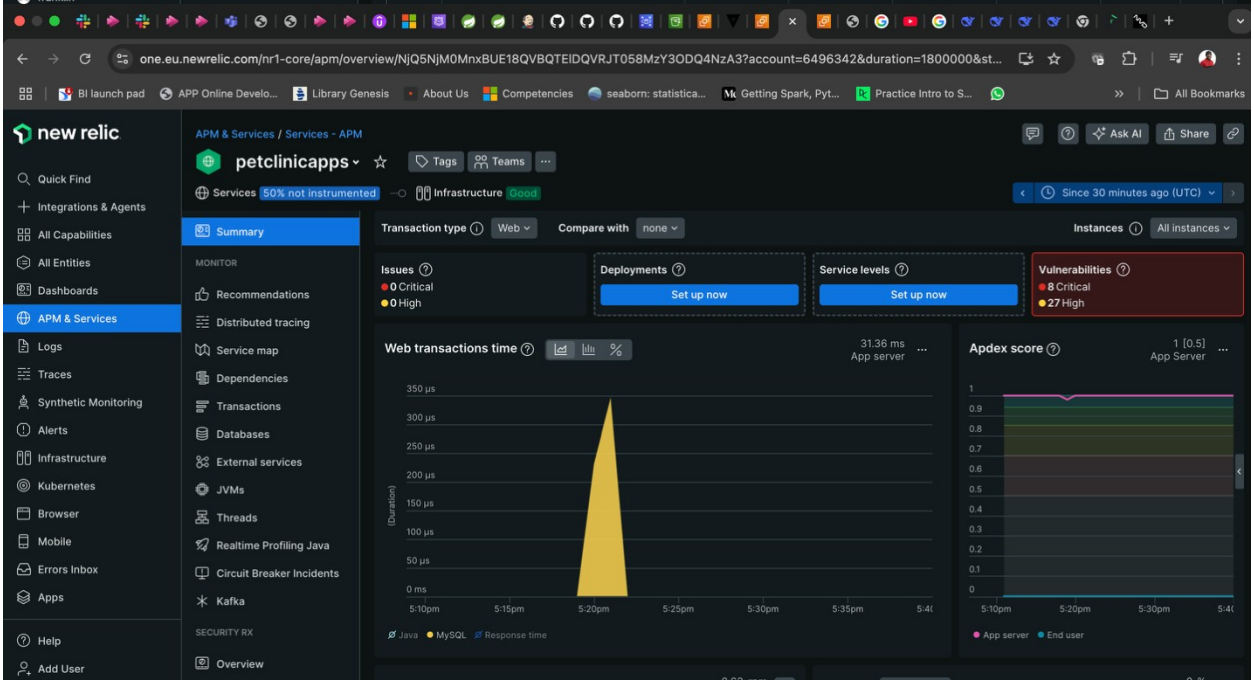


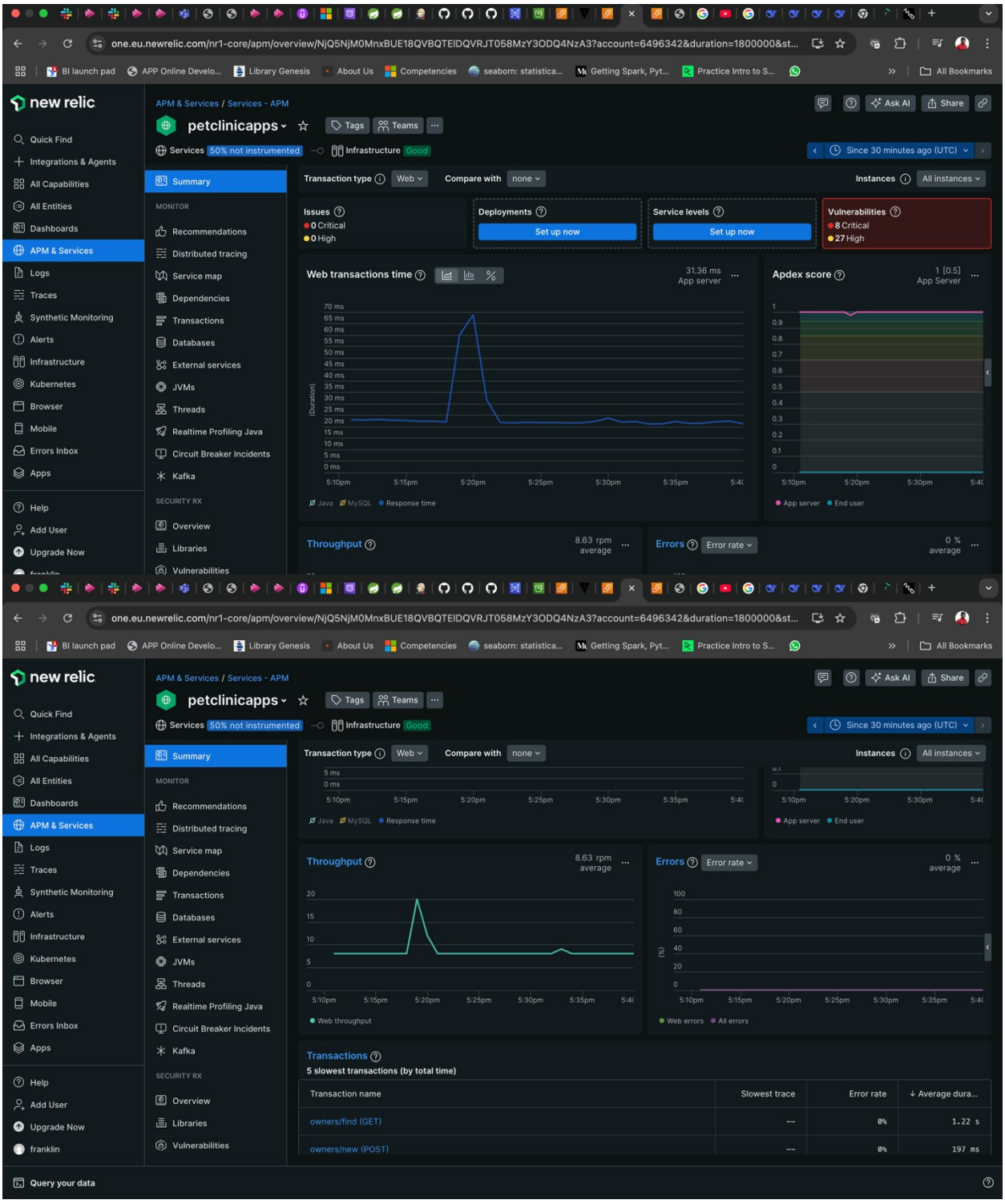
see throughput

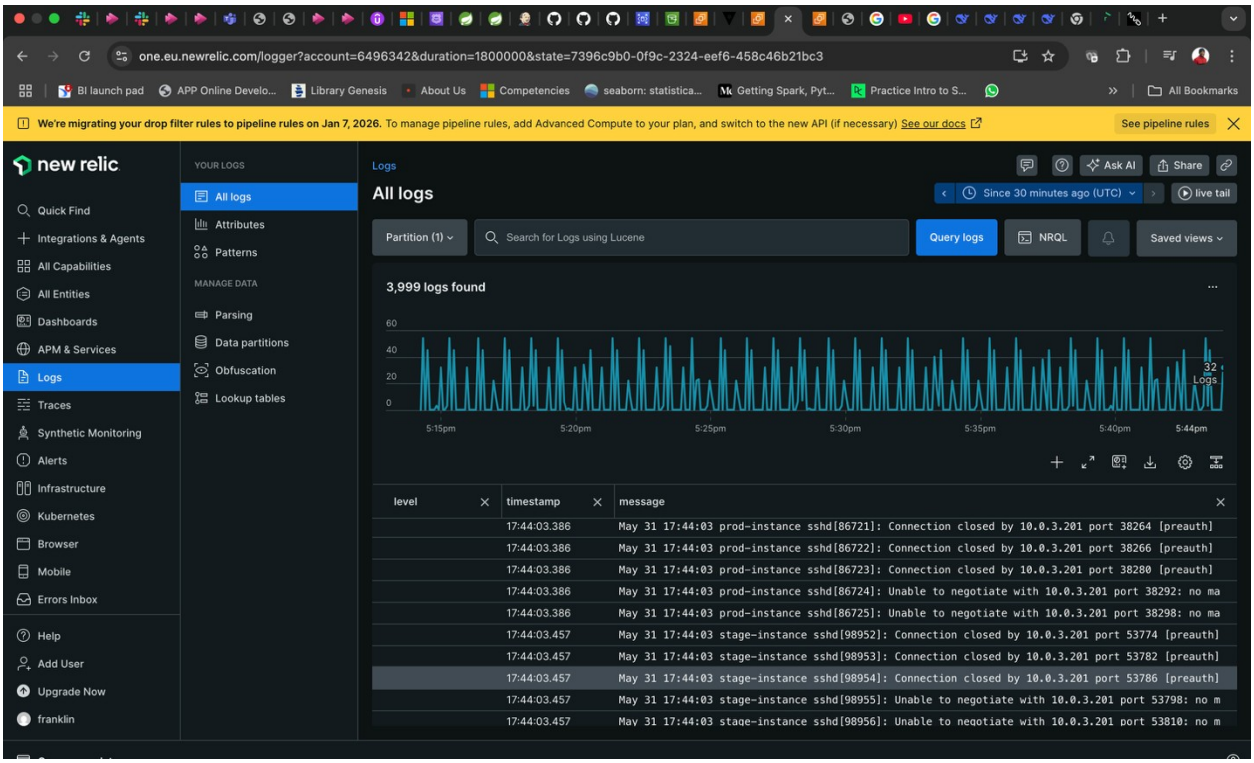


see infrastructure monitoring


one.eu.newrelic.com/infra/infrastructure/hosts?account=6496342&begin=1748706019000&end=1748713219000&state=d69b0957-6300-822a-01d...
new relic
Integrations & Agents
All Capabilities
All Entities
Dashboards
APM & Services
Logs
Traces
Synthetic Monitoring
Alerts
Infrastructure
Kubernetes
Browser
Mobile
Errors Inbox
Apps
Help
Add User
Upgrade Now
franklin
OVERVIEW
Hosts
Kubernetes
Databases
New
Network
Inventory
Events
INTEGRATIONS
AWS
GovCloud
Azure
GCP
Third-party services
SETTINGS
Alerts
Infrastructure
Hosts
Overview
Classic
List
Navigator
Lookout
from May 31, 03:40 pm to May 31, 05:40 pm (UTC)
Share your feedback
Ask AI
Share
Metrics
CPU (%)
Memory used (%)
Disk used (%)
Disk utilization (%)
4:00pm
5:00pm
Summary
View
Name
Applications
Agent version
CPU usage (%)
Memory usa...
Storage usag
ansible-server
1.64.0
8.93%
18.23%
21.2
prod-instance
1.64.0
3.79%
29.14%
34.0
stage-instance
1.64.0
4.89%
29.58%
34.4


one.eu.newrelic.com/nr1-core/apm/overview/NjQ5NjM0MnxBUE18QVBQTElDQVRJT058MzY3ODQ4NzA3?account=6496342&duration=1800000&st...
new relic
Quick Find
Integrations & Agents
All Capabilities
All Entities
Dashboards
APM & Services
Logs
Traces
Synthetic Monitoring
Alerts
Infrastructure
Kubernetes
Browser
Mobile
Errors Inbox
Apps
Help
Add User
APM & Services / Services - APM
petclinicapps
Tags
Teams
Services 50% not instrumented
Infrastructure Good
Since 30 minutes ago (UTC)
Ask AI
Share
Summary
MONITOR
Recommendations
Distributed tracing
Service map
Dependencies
Transactions
Databases
External services
JVMs
Threads
Realtime Profiling Java
Circuit Breaker Incidents
Kafka
SECURITY RX
Overview
Transaction type
Web
Compare with
none
Instances
All instances
Issues
0 Critical
0 High
Deployments
Set up now
Service levels
Set up now
Vulnerabilities
8 Critical
27 High
Web transactions time
31.36 ms
App server
Apdex score
1 [0.5]
App Server
(Duration)
Java
MySQL
Response time
App server
End user


one.eu.newrelic.com/nr1-core/apm/overview/NjQ5NjM0MnxBUE18QVBQTElDQVRJT058MzY3ODQ4NzA3?account=6496342&duration=1800000&st...
new relic
APM & Services / Services - APM
petclinicapps
Tags
Teams
Services 50% not instrumented
Infrastructure Good
Since 30 minutes ago (UTC)
Ask AI
Share
Summary
MONITOR
Recommendations
Distributed tracing
Service map
Dependencies
Transactions
Databases
External services
JVMs
Threads
Realtime Profiling Java
Circuit Breaker Incidents
Kafka
SECURITY RX
Overview
Libraries
Vulnerabilities
Quick Find
Integrations & Agents
All Capabilities
All Entities
Dashboards
APM & Services
Logs
Traces
Synthetic Monitoring
Alerts
Infrastructure
Kubernetes
Browser
Mobile
Errors Inbox
Apps
Help
Add User
Upgrade Now
franklin
Transaction type Web
Compare with none
Instances All instances
Issues
0 Critical
0 High
Deployments
Set up now
Service levels
Set up now
Vulnerabilities
8 Critical
27 High
Web transactions time
31.36 ms
App server
Apdex score
1 [0.5]
App Server
Throughput
8.63 rpm
average
Errors
Error rate
0 %
average
Java
MySQL
Response time
App server
End user
Web throughput
Web errors
All errors
Transactions
5 slowest transactions (by total time)
Transaction name
Slowest trace
Error rate
↓ Average dura...
owners/find (GET)
owners/new (POST)
1.22 s
197 ms
Query your data



Further improvement:
Working to automate access key for vault to be copied automatically into the vault provider file
tighten up security by shutting ssh port and granting access through SSM only

Conclusion
In conclusion, automating application deployment is essential for organizations to meet evolving market demands efficiently. By streamlining deployment processes, businesses can accelerate delivery, enhance productivity, and ultimately drive greater profitability.