**Geist-UI CI/CD**

Prerequisites:

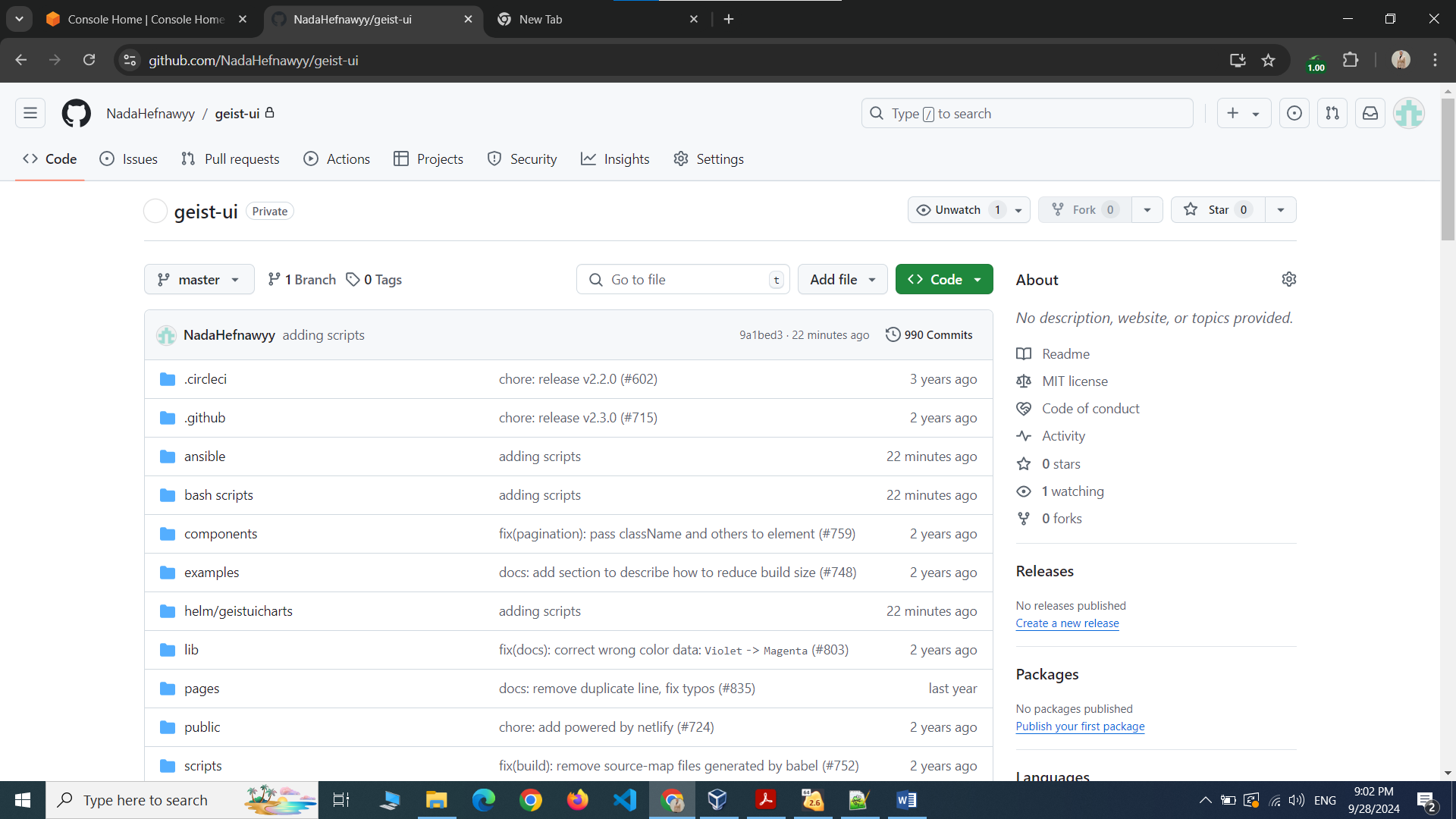
* Terraform
* Ansible (Optional we can SSH to the instances and run the scripts)
* AWS Account
* DockerHub Account
* Domain Name

Tools:

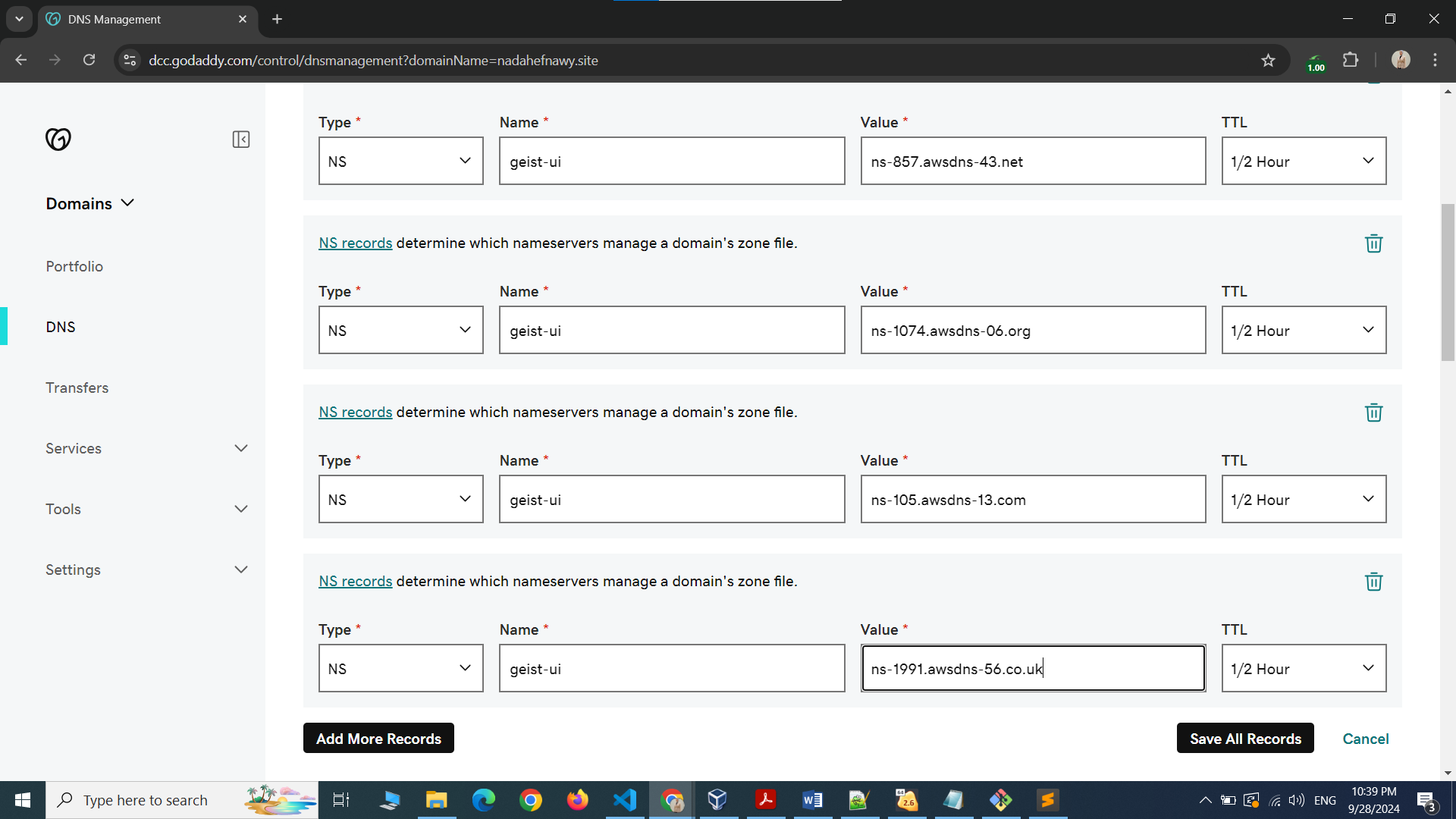
* Jenkins
* Docker
* Terraform
* Ansible
* Prometheus
* Helm
* Github
* AWS (EC2 – S3 Bucket – Route53 - IAM)

**Step 1:**

I cloned the geist-ui library repository form GitHub then created a new private repo on my account and pushed the project to it



I also added my Route53 NS records to my DNS

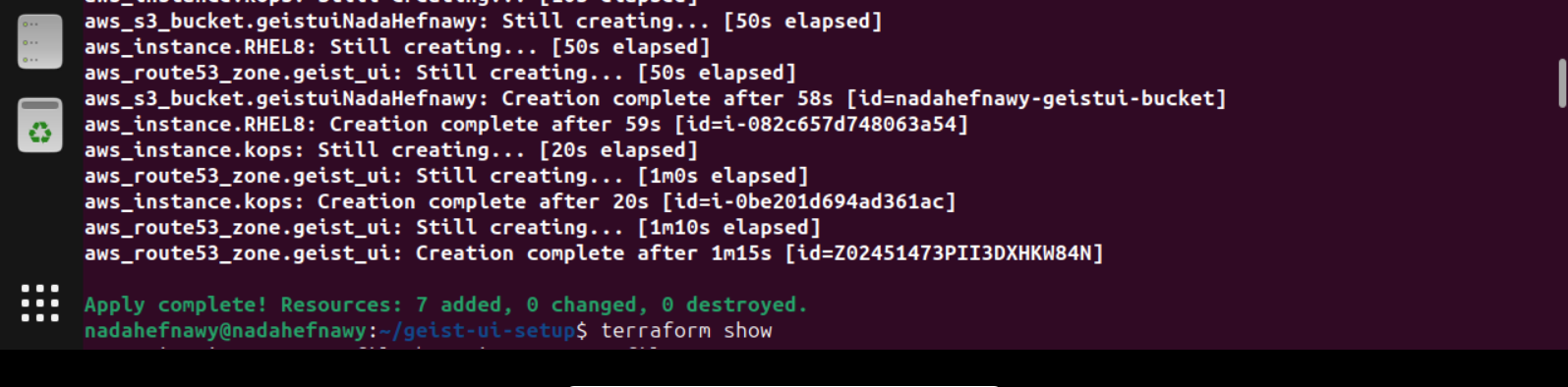


**Step 2:**

I will then open my Linux machine and run the Terraform commands that will create the infrastructure of the project

It will create:

* 1 RHEL 8 ec2 instance that will be the Jenkins server
* 1 ec2 instance that we will use to setup the Kubernetes cluster using Kops
* Route53 hosted zone to manage DNS for the Kubernetes cluster
* S3 bucket to store the Kubernetes cluster's state and configuration
* IAM Role to attach to the kops instance to be able to use AWS-CLI



**Step 3:**

I will update the ip addresses in the Ansible Inventory file in order to SSH successfully

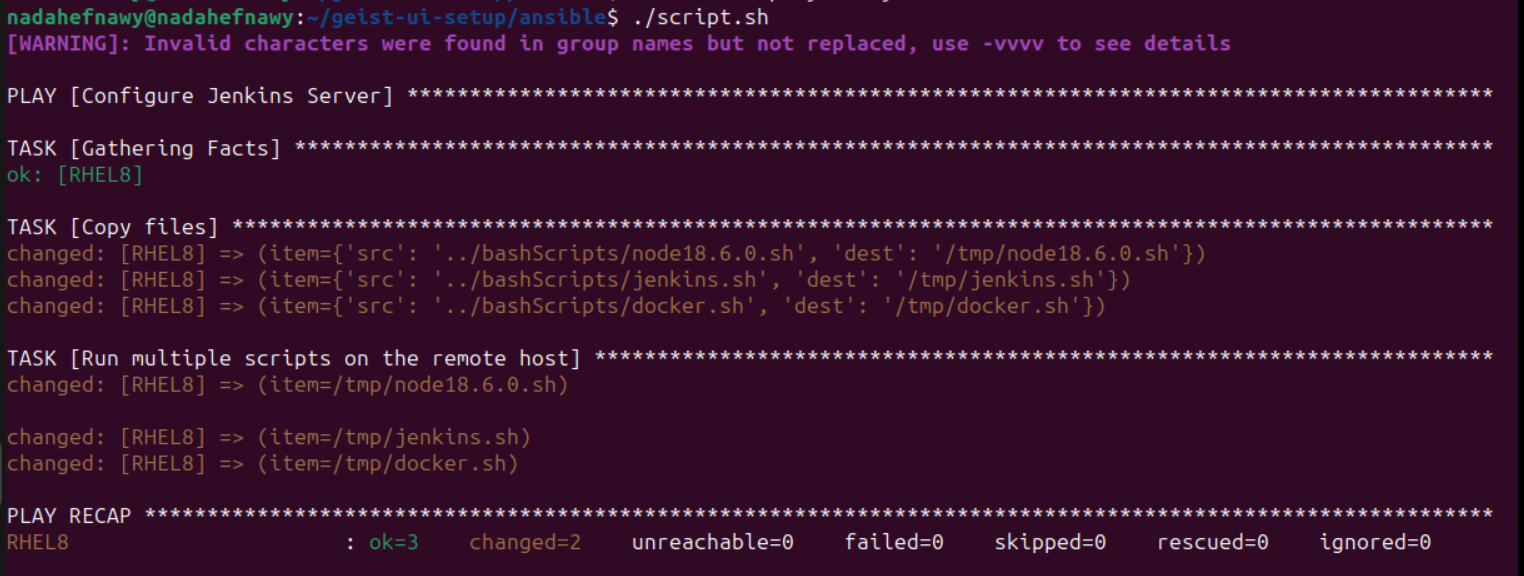
I will run Ansible commands that will run the playbooks that will configure the ec2 instances

For the Jenkins servers I wrote several bash scripts that will configure everything without any human intervention as following:

* Install Node version 18.6.0 and install yarn
* Install Jenkins and start its service
* Install Docker and start its service
* Add the Jenkins user to the Docker group in order to allow Jenkins to run Docker commands
* Cat the Jenkins password in order to be able to login to Jenkins

For the kops machine there are also bash scripts that I ran using Ansible as following:

* Install AWS-CLI
* Install Kubectl
* Install Kops
* Setup Kubernetes cluster using Kops
* Validate the cluster creation
* Install Helm
* Install Prometheus using Helm
* Install Java and configure directory on Kops to run the kops instance as a slave in Jenkins



**Step 4:**

Now I need to write my Dockerfile and push it to DockerHub I used a node:18.6.0-alpine as the base image and I am not running the command “yarn” here as it will increase the size of the image so I am just copying the library code with its dependencies after running “yarn” command from within the Jenkins server in order to reduce the image size (It will all be clear in the Jenkinsfile)

FROM node:18.6.0-alpine  
  
# Set memory limit for Node.js  
ENV NODE\_OPTIONS="--max-old-space-size=4096"  
  
# Step 2: Set the working directory inside the container  
WORKDIR /app  
# Step 5: Copy the rest of the library code  
COPY . .  
  
# Step 6: Expose the library's port (e.g., 8001)  
EXPOSE 3000  
  
# Step 7: Define the command to run your application  
CMD ["yarn", "dev"]

**Step 5:**

Now I have my infrastructure ready and configured and my Jenkins server up and running so I need to configure it as following:

* Install required plugins
* Add needed credentials (DockerHub - GitHub)
* Write my pipeline code and push it to GitHub
* Connect Jenkins to the GitHub repository
* Create a Webhook in GitHub to use in Jenkins to trigger any change in the code