

Project 2 React Application Deployment

(To Deploy the given React application to a production ready state)

Application:

Clone the below mentioned repository and deploy the application (Run application in port 3000).

Repo URL : <https://github.com/Vennilavan12/Trend.git>

Docker:

- Dockerize the application by creating Dockerfile
- Build an application and check output using docker image.

Terraform:

- Define infrastructure in main.tf to create VPC, IAM, EC2 with Jenkins, etc.
- Use terraform command to provision infrastructure.

DockerHub:

- Create a DockerHub repository.

Kubernetes:

- Setup Kubernetes in AWS EKS and Confirm EKS cluster is running.
- Write deployment and service YAML files.
- Deploy using kubectl via Jenkins.

Version Control:

- Push the codebase to a Git provider (GitHub).
- Add gitignore and dockerignore files and use CLI commands to push code.

Jenkins:

- Install Jenkins and necessary plugins (Docker, Git, Kubernetes, Pipeline) for build, push & deploy applications.
- Setup Github and jenkins integration using github webhook build trigger for auto build for every commit.
- Create a declarative pipeline script and pipeline project to build, push & deploy using CI-CD.

Monitoring:

- Setup a monitoring system to check the health of the cluster or application (opensource) is highly appreciable.

Step 1: Local Project Setup and Version Control

1. Create a new folder for your project (e.g., 'Project2'DevOps')
2. Clone the original repository into this folder: `git clone https://github.com/Vennilavan12/Trend.git`
3. Navigate into the newly cloned project folder: `cd Trend`
4. First, check the current remote URL (it should show 'Vennilavan12/Trend.git'): `git remote -v`
5. Now, set the new remote URL to your repository : `git remote set-url origin https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git`
6. Verify that the remote URL has been updated to your repository: `git remote -v`
7. Change the default branch name to 'main' for consistency (if it's not already): `git branch -M main`
8. Push the existing code from your local machine to your new GitHub repository: `git push -u origin main`
9. Add .gitignore and .dockerignore Files
10. Commit and Push the New Files

- ***.gitignore***

```
# Node modules
node_modules
.env

# Build artifacts
build
dist
*.log

# OS generated files
.DS_Store
.idea
.vscode
```

- ***.dockerignore***

```
node_modules
.git
.gitignore
Dockerfile
Jenkinsfile
README.md
.env
```

- `git commit -m "Add .gitignore and .dockerignore files"`
- `git push origin main`

Step 2: Dockerize the Application

In this step, you will create a Docker image of your React application. This image will contain a web server (Nginx) that is configured to serve your application's static files from the dist folder. This makes your application portable and runnable in any environment that supports Docker.

2.1. Create the Dockerfile

```
FROM nginx:alpine  
  
COPY nginx.conf /etc/nginx/conf.d/default.conf  
  
COPY dist /usr/share/nginx/html  
  
EXPOSE 3000  
  
CMD ["nginx", "-g", "daemon off;"]
```

2.2. Create the nginx.conf

```
server {  
  
    listen 3000;  
  
    server_name localhost;  
  
    location / {  
  
        root /usr/share/nginx/html;  
  
        index index.html index.htm;  
  
        try_files $uri $uri/ /index.html;  
  
    }  
  
    error_page 500 502 503 504 /50x.html;  
  
    location = /50x.html {  
  
        root /usr/share/nginx/html;  
  
    }  
}
```

2.3. Build and Test the Docker Image Locally

```
docker build -t trend-react-app:latest .  
  
docker images
```

```
docker run -d -p 3000:3000 --name trend-app-container trend-react-app:latest
```

```
docker stop trend-app-container
```

```
docker rm trend-app-container
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure named "PROJECT2_DEVOPS" containing a "Trend" folder with "dist", "assets", "index.html", "vite.svg", ".dockerignore", ".gitignore", "Dockerfile", and "nginx.conf".
- Terminal:** Displays the following Docker commands and their output:
 - `PS P:\Project2_DevOps\Trend> docker images`
REPOSITORY TAG IMAGE ID CREATED SIZE
trend-react-app latest 758da245c51b 13 seconds ago 97.6MB
 - `PS P:\Project2_DevOps\Trend> docker run -d -p 3000:3000 --name trend-app-container trend-react-app:latest`
2eca9ea5ae11abbda67ea16c286f21b7ad2776f9c738122ea9dab07fab4d340
 - `PS P:\Project2_DevOps\Trend> docker ps`
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
2eca9ea5ae11 trend-react-app:latest "/docke... 41 seconds ago Up 40 seconds 0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp trend-app-container
- Bottom Status Bar:** Shows the current weather (24°C, Mostly cloudy), system icons, and the date/time (02-08-2025, 19:18).

The screenshot shows a web browser window displaying the Trendify website at `127.0.0.1:3000`. The page content includes:

- Trendify Logo:** Find What Moves You
- Navigation:** HOME, COLLECTION, ABOUT, CONTACT
- Section Headers:** OUR BEST SELLERS, Latest Arrivals
- Call-to-Action:** SHOP NOW
- Image:** A large image of a woman with blonde hair wearing a black turtleneck.
- Bottom Status Bar:** Shows the current weather (24°C, Mostly cloudy), system icons, and the date/time (02-08-2025, 19:19).

2.4. Commit and Push Docker Files

```
git add Dockerfile nginx.conf
```

```
git commit -m "Add Dockerfile and Nginx config for application deployment"
```

```
git push origin main
```

The screenshot shows the Visual Studio Code interface with the following terminal history:

```
Project2_DevOps Trend > nginx.conf
1 server {
2   listen 3000;
3   server_name localhost;
4

-> => unpacking to docker.io/library/trend-react-app:latest
PS P:\Project2_DevOps\Trend> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
trend-react-app    latest   758da245c51b  13 seconds ago  97.6MB
PS P:\Project2_DevOps\Trend> docker run -d -p 3000:3000 --name trend-app-container trend-react-app:latest
2eca9ea5ae11abbda67ea16c286f21b7ad2776f9c738122ea9dab07fab4d340
PS P:\Project2_DevOps\Trend> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED          STATUS           PORTS
2eca9ea5ae11        trend-react-app:latest "/docker-entrypoint..."  41 seconds ago   Up 40 seconds   0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp   NAMES
ainer
PS P:\Project2_DevOps\Trend> git add Dockerfile nginx.conf
PS P:\Project2_DevOps\Trend> git commit -m "Add Dockerfile and Nginx config for application deployment"
[main bbfa7e8] Add Dockerfile and Nginx config for application deployment
 2 files changed, 34 insertions(+)
   create mode 100644 Dockerfile
   create mode 100644 nginx.conf
PS P:\Project2_DevOps\Trend> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1006 bytes | 1006.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git
 1644306..bbfa7e8  main -> main
PS P:\Project2_DevOps\Trend >
```

The terminal also shows the current working directory as PS P:\Project2_DevOps\Trend. The status bar at the bottom indicates the user is PranuthHM (now) and the date is 02-08-2025.

The screenshot shows a GitHub repository page for "React_Application_Deployment_Document_2_Trend". The commit history is as follows:

- bbfa7e8 - 1 minute ago (PranuthHM): Add Dockerfile and Nginx config for application deployment
- 2 months ago (dist): Config Files
- 1 hour ago (.dockerignore): Add .gitignore and .dockerignore files
- 1 hour ago (.gitignore): Add .gitignore and .dockerignore files
- 1 minute ago (Dockerfile): Add Dockerfile and Nginx config for application deployment
- 1 minute ago (nginx.conf): Add Dockerfile and Nginx config for application deployment

The repository has 3 commits, 0 branches, and 0 tags. The "About" section describes the repository as "Production-ready deployment of a React application utilizing Docker for containerization, Terraform for infrastructure as code, Kubernetes for orchestration, Jenkins for CI/CD automation, and monitoring with Prometheus and Grafana, all hosted on AWS Cloud." The "Activity" section shows 0 stars, 0 watching, and 0 forks. The status bar at the bottom indicates the weather is 23°C Rain showers and the date is 02-08-2025.

Step 3: Terraform Infrastructure

3.1. Create a Terraform Directory and main.tf

1. # Go back to the parent directory (e.g., DevOpsProject) : cd ..

2. # Create a new directory for Terraform files

mkdir terraform

cd terraform

3. Inside this new terraform directory, create a file named: main.tf

3.2. Define Your Infrastructure in main.tf

```
terraform {
```

```
  required_providers {
```

```
    aws = {
```

```
      source = "hashicorp/aws"
```

```
      version = "~> 4.0"
```

```
    }
```

```
}
```

```
# Remote state storage in S3
```

```
  backend "s3" {
```

```
    bucket = "your-unique-terraform-state-bucket" # ⚠ REPLACE THIS NAME
```

```
    key    = "devops-project/terraform.tfstate"
```

```
    region = "ap-south-1"
```

```
}
```

```
}
```

```
  provider "aws" {
```

```
    region = "ap-south-1"
```

```
}
```

```

# --- VPC and Subnet ---

resource "aws_vpc" "project_vpc" {
    cidr_block      = "10.0.0.0/16"
    enable_dns_support = true
    enable_dns_hostnames = true

    tags = {
        Name = "Project-VPC"
    }
}

resource "aws_subnet" "public_subnet" {
    vpc_id          = aws_vpc.project_vpc.id
    cidr_block      = "10.0.1.0/24"
    availability_zone = "ap-south-1a"
    map_public_ip_on_launch = true

    tags = {
        Name = "Project-Public-Subnet"
    }
}

resource "aws_subnet" "public_subnet_2" {
    vpc_id          = aws_vpc.project_vpc.id
    cidr_block      = "10.0.2.0/24" # New CIDR block for the second subnet
    availability_zone = "ap-south-1b" # Different AZ
    map_public_ip_on_launch = true
}

```

```
tags = {  
    Name = "Project-Public-Subnet-2"  
}  
}  
  
resource "aws_internet_gateway" "project_igw" {  
    vpc_id = aws_vpc.project_vpc.id  
  
    tags = {  
        Name = "Project-IGW"  
    }  
}  
  
resource "aws_route_table" "project_rt" {  
    vpc_id = aws_vpc.project_vpc.id  
  
    route {  
        cidr_block = "0.0.0.0/0"  
        gateway_id = aws_internet_gateway.project_igw.id  
    }  
  
    tags = {  
        Name = "Project-RouteTable"  
    }  
}  
  
resource "aws_route_table_association" "project_rta" {  
    subnet_id      = aws_subnet.public_subnet.id
```

```

route_table_id = aws_route_table.project_rt.id
}

# --- Security Group for Jenkins EC2 ---

resource "aws_security_group" "jenkins_sg" {
  vpc_id      = aws_vpc.project_vpc.id
  name        = "jenkins-ec2-sg"
  description = "Security group for Jenkins EC2 instance"

  ingress {
    from_port  = 22
    to_port    = 22
    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"]
  }

  ingress {
    from_port  = 50000
    to_port    = 50000
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

```

```
}
```

```
egress {
```

```
    from_port = 0
```

```
    to_port = 0
```

```
    protocol = "-I"
```

```
    cidr_blocks = ["0.0.0.0/0"]
```

```
}
```

```
tags = {
```

```
    Name = "Jenkins-EC2-SG"
```

```
}
```

```
}
```

```
# --- IAM Role for Jenkins EC2 ---
```

```
resource "aws_iam_role" "ec2_instance_profile_role" {
```

```
    name = "ec2-jenkins-role"
```

```
    assume_role_policy = jsonencode({
```

```
        Version = "2012-10-17"
```

```
        Statement = [
```

```
{
```

```
            Action = "sts:AssumeRole"
```

```
            Effect = "Allow"
```

```
            Principal = {
```

```
                Service = "ec2.amazonaws.com"
```

```
}
```

```
,
```

```

    ]
}

tags = {
  Name = "EC2-Jenkins-Role"
}

}

resource "aws_iam_role_policy_attachment" "ec2_policy_attachment" {
  role      = aws_iam_role.ec2_instance_profile_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryFullAccess"
}

resource "aws_iam_role_policy_attachment" "eks_cluster_policy" {
  role      = aws_iam_role.ec2_instance_profile_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
}

resource "aws_iam_role_policy_attachment" "eks_service_policy" {
  role      = aws_iam_role.ec2_instance_profile_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
}

resource "aws_iam_instance_profile" "ec2_instance_profile" {
  name = "ec2-jenkins-instance-profile"
  role = aws_iam_role.ec2_instance_profile_role.name
}

```

```

# --- SSH Key Pair Generation ---

resource "tls_private_key" "sshkey" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "local_file" "private_key" {
  content      = tls_private_key.sshkey.private_key_pem
  filename     = "${path.module}/terraform_ssh_key.pem"
  file_permission = "0400"
}

resource "aws_key_pair" "generated" {
  key_name  = "terraform-jenkins-key"
  public_key = tls_private_key.sshkey.public_key_openssh
}

# --- Jenkins EC2 Instance ---

resource "aws_instance" "jenkins_ec2" {
  ami           = "ami-00bb6a80f01f03502" # Ubuntu Server 22.04 LTS (HVM)
  instance_type = "t3.medium"
  key_name      = aws_key_pair.generated.key_name
  subnet_id     = aws_subnet.public_subnet.id
  vpc_security_group_ids = [aws_security_group.jenkins_sg.id]
  associate_public_ip_address = true
  iam_instance_profile = aws_iam_instance_profile.ec2_instance_profile.name
}

```

```
user_data = <<-EOF

#!/bin/bash

sudo apt update -y

sudo apt upgrade -y


# Install Java 17

sudo apt install -y openjdk-17-jdk


# Add Jenkins repository key

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian binary/" | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update -y

sudo apt install -y jenkins


# Install Docker

sudo apt install -y docker.io

sudo usermod -aG docker jenkins

sudo usermod -aG docker ubuntu

sudo systemctl enable docker

sudo systemctl start docker


# Start and enable Jenkins

sudo systemctl start jenkins

sudo systemctl enable jenkins
```

```

# Install kubectl

curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl


# Install eksctl

curl -LO
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz"
tar -xzf eksctl_Linux_amd64.tar.gz
sudo mv eksctl /usr/local/bin


# Install AWS CLI

sudo apt install awscli -y

EOF

tags = {

Name = "Jenkins-EC2-Instance"

}

output "jenkins_public_ip" {

description = "The public IP address of the Jenkins EC2 instance"

value      = aws_instance.jenkins_ec2.public_ip

}

output "ssh_private_key_path" {

description = "Path to the generated SSH private key"

value      = "${path.module}/terraform_ssh_key.pem"

}

```

3.3. Run Terraform Commands

- `terraform init`
- `terraform plan -out tfplan`
- `terraform apply "tfplan"`

```

resource "aws_internet_gateway" "project_igw" {
    Name = "Project-IGW"
}

aws_route_table_association.project_rta: Creation complete after 0s [id=rtbassoc-02ebba68a8c9fd77f]
tls_private_key.sshkey: Still creating... [00m30s elapsed]
tls_private_key.sshkey: Still creating... [00m40s elapsed]
tls_private_key.sshkey: Still creating... [00m50s elapsed]
tls_private_key.sshkey: Still creating... [01m00s elapsed]
tls_private_key.sshkey: Creation complete after 1m5s [id=c96e51e276ad7703ef29350bad3da136db2fa4f]
aws_key_pair.generated: Creating...
local_file.private_key: Creating...
local_file.private_key: Creation complete after 0s [id=50d252e373d8234cd4a349ba2f31b388f7469dfb]
aws_key_pair.generated: Creation complete after 0s [id=terraform-jenkins-key]
aws_instance.jenkins_ec2: Creating...
aws_instance.jenkins_ec2: Still creating... [00m10s elapsed]
aws_instance.jenkins_ec2: Creation complete after 13s [id=i-07f3538c6370bc182]

Apply complete! Resources: 15 added, 0 changed, 0 destroyed.

Outputs:

jenkins_public_ip = "13.235.95.105"
ssh_private_key_path = "./terraform_ssh_key.pem"

```

Name	Instance ID	Instance state	Instance type	Status check	Availability Zone	Public IPv4 DNS
Jenkins-EC2-Instance	i-07f3538c6370bc182	Running	t3.medium	3/3 checks passed	ap-south-1a	ec2-13-235-95-105.ap...

Name	AWS Region	Creation date
codepipeline-ap-south-1- eb3ab312d5cf-4c3f-af36- 01fddc79a528	Asia Pacific (Mumbai) ap-south-1	June 14, 2025, 21:40:52 (UTC+05:30)
pranuth-terraform-state-bucket	Asia Pacific (Mumbai) ap-south-1	August 3, 2025, 13:38:31 (UTC+05:30)

vpcs | VPC Console

ap-south-1.console.aws.amazon.com/vpcconsole/home?region=ap-south-1#vpcs:

AWS Search [Alt+S]

EC2 VPC S3 IAM CloudTrail CloudWatch CodePipeline CodeDeploy CodeBuild

VPC Your VPCs

VPC dashboard EC2 Global View Filter by VPC

Virtual private cloud Your VPCs Subnets Route tables Internet gateways Egress-only internet gateways DHCP option sets Elastic IPs Managed prefix lists NAT gateways Peering connections

Security Network ACLs Security groups

PrivateLink and Lattice

Select a VPC above

Your VPCs (2) Info

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set
Project-VPC	vpc-0162b907e18230586	Available	Off	10.0.0.0/16	-	dopt-0adff67cad
-	vpc-0f0aedf504da86e4a	Available	Off	172.31.0.0/16	-	dopt-0adff67cad

Last updated less than a minute ago Actions Create VPC

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

27°C Mostly cloudy

Search

14:05 03-08-2025

Step 4: Jenkins Setup and Configuration (Jenkins server configuration and integration with GitHub and DockerHub.)

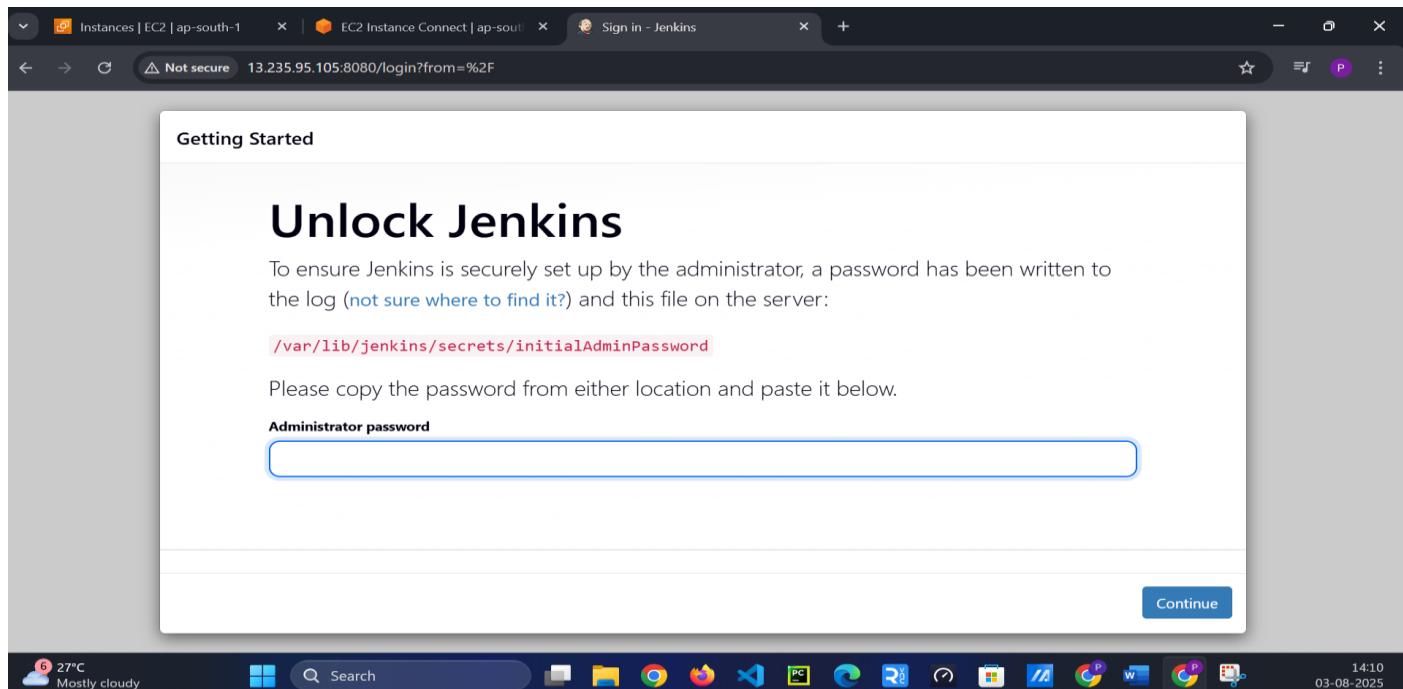
4.1. Access Jenkins and Complete Initial Setup

Find Your Jenkins Public IP:

Access the Jenkins UI:

Retrieve the Initial Admin Password: `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

Complete the Jenkins Wizard:



```
System information as of Sun Aug  3 08:37:57 UTC 2025
System load: 0.09      Temperature: -273.1 C
Usage of /: 52.7% of 6.71GB  Processes: 116
Memory usage: 24%
Swap usage: 0%          IPv4 address for ens5: 10.0.1.209

expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-1-209:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
n3h241a0s#04538af53bf09c5f099bf
ubuntu@ip-10-0-1-209:~$ ^C
ubuntu@ip-10-0-1-209:~$ ^C

I-07f3538c6370bc182 (Jenkins-EC2-Instance)
PublicIPs: 13.235.95.105 PrivateIPs: 10.0.1.209

CloudShell Feedback
6 27°C Mostly cloudy Search
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
14:09 03-08-2025
```

The screenshot shows the Jenkins 'Getting Started' page. At the top, there's a table of available plugins:

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✗ Credentials Binding	✗ commons-lang3 v3.x Jenkins API
✗ Timestamper	✗ Workspace Cleanup	✗ Ant	✗ Gradle	✗ Ionicons API
✗ Pipeline	✗ GitHub Branch Source	✗ Pipeline: GitHub Groovy Libraries	✗ Pipeline Graph View	✗ ASM API
✗ Git	✗ SSH Build Agents	✗ Matrix Authorization Strategy	✗ LDAP	✗ JSON Path API
✗ Email Extension	✗ Mailer	✗ Dark Theme		✗ Structs

On the right side, there's a sidebar with more plugin details:

- Folders
 - OWASP Markup Formatter
 - ASM API
 - JSON Path API
 - Structs
 - Pipeline: Step API
 - commons-text API
 - Text Macro
- Build Pipeline
 - bouncycastle API
 - Credentials
 - Plain Credentials
 - Variant

Jenkins 2.521

The screenshot shows the Jenkins Dashboard. On the left, there are navigation links for 'New Item', 'Build History', 'Build Queue' (which shows 'No builds in the queue.'), and 'Build Executor Status' (which shows '0/2'). In the center, there's a 'Welcome to Jenkins!' message and a 'Start building your software project' section with a 'Create a job' button. Below that is a 'Set up a distributed build' section with links for 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'. At the bottom, there's a status bar showing weather (27°C, Mostly cloudy), system icons, and the date/time (14:14 03-08-2025). REST API and Jenkins 2.521 are also mentioned.

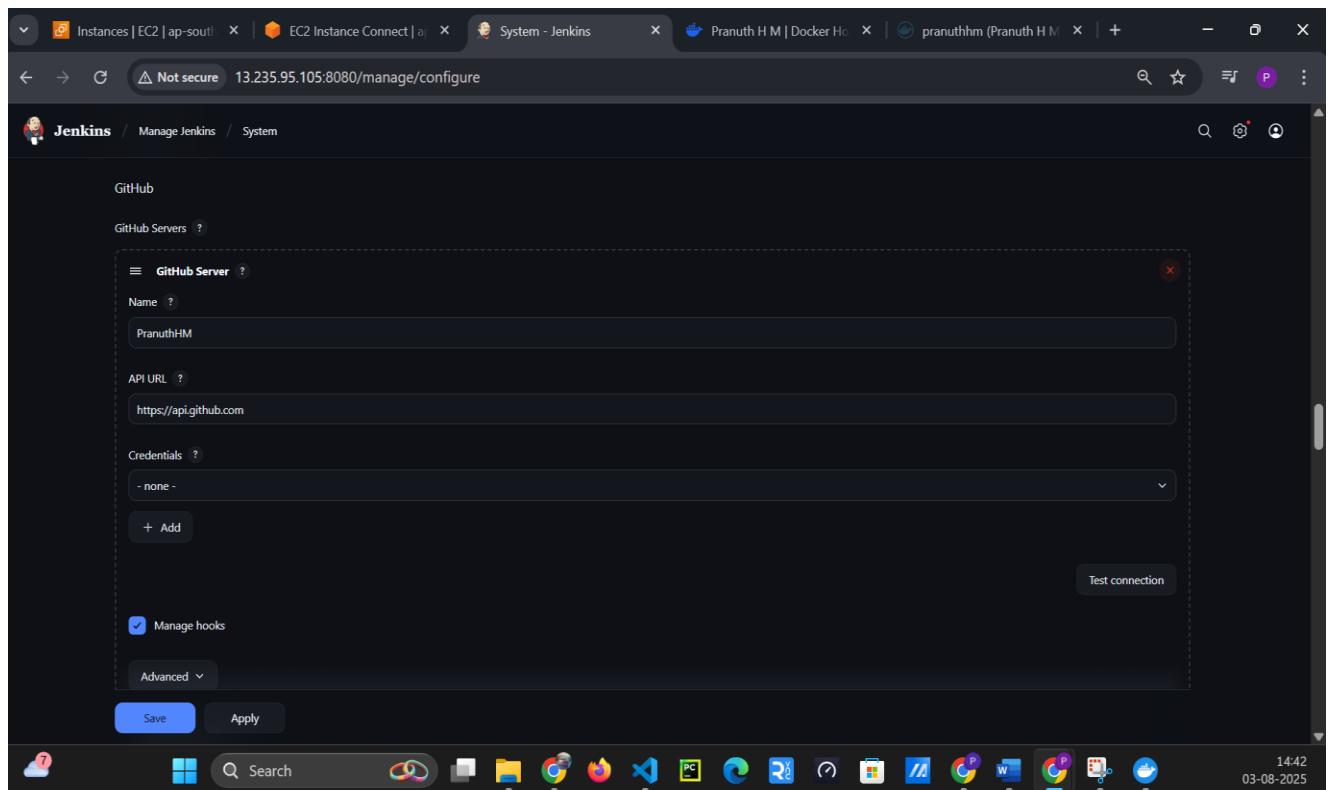
4.2. Install Necessary Plugins: go to Manage Jenkins -> Manage Plugins.-> Click on the Available tab.

- GitHub Plugin
- GitHub Branch Source Plugin
- Docker Pipeline
- Kubernetes CLI

4.3. Configure GitHub Webhook Trigger

4.3.1 In Jenkins:

- Go to Manage Jenkins -> Configure System.
- Scroll down to the GitHub section.
- Click Add GitHub Server and select GitHub Server.
- Click Manage Hooks and ensure the box Manage hooks for this server is checked.
- Click Save.



4.3.2 In GitHub:

- Go to your repository:
https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git
- Click on Settings -> Webhooks->Click Add webhook.
- **Payload URL:** Enter http://<YOUR_JENKINS_PUBLIC_IP>:8080/github-webhook/
- **Content type:** Select application/json.
- **Which events would you like to trigger this webhook?:** Select Just the push event.
- Make sure Active is checked.
- Click Add webhook.

The screenshot shows the GitHub 'Webhooks' settings page for a repository named 'React_Application_Deployment_Document_2_Trend'. The left sidebar has 'Webhooks' selected under 'Code and automation'. The main area displays a single webhook entry:

- URL: <http://13.235.95.105:8080/github-webhook/push> (push)
- Status: This hook has never been triggered.
- Buttons: 'Edit' and 'Delete'

At the top, a message says: 'Okay, that hook was successfully created. We sent a ping payload to test it out! Read more about it at <https://docs.github.com/webhooks/#ping-event>'.

4.4. Store DockerHub Credentials in Jenkins:

- In Jenkins, go to Manage Jenkins -> Manage Credentials.
- Click on (global) -> Add Credentials.
- From the dropdown, select Username with password.
- Username: Your DockerHub username.
- Password: Your DockerHub password.
- ID: dockerhub_credentials (This ID is very important; you will use it in your Jenkinsfile later).
- Description: DockerHub Credentials
- Click Create.

The screenshot shows the Jenkins 'Manage Jenkins / Credentials' page. A single credential is listed:

T	P	Store	ID	Name
		System	(global)	pranuthhm/*****

Below the table, a section titled 'Stores scoped to Jenkins' shows a single store entry:

P	Store	Domains
	System	(global)

At the bottom, there are icons for Icon: S M L and links for REST API and Jenkins 2.521.

Step 5: DockerHub Repository: This is a quick but crucial step. Your Jenkins pipeline will build a Docker image of your React application and then push that image to a repository. You need to create this repository on DockerHub first.

1. Navigate to DockerHub:

- Open your web browser and go to <https://hub.docker.com/>.
- Log in to your DockerHub account if you haven't already.

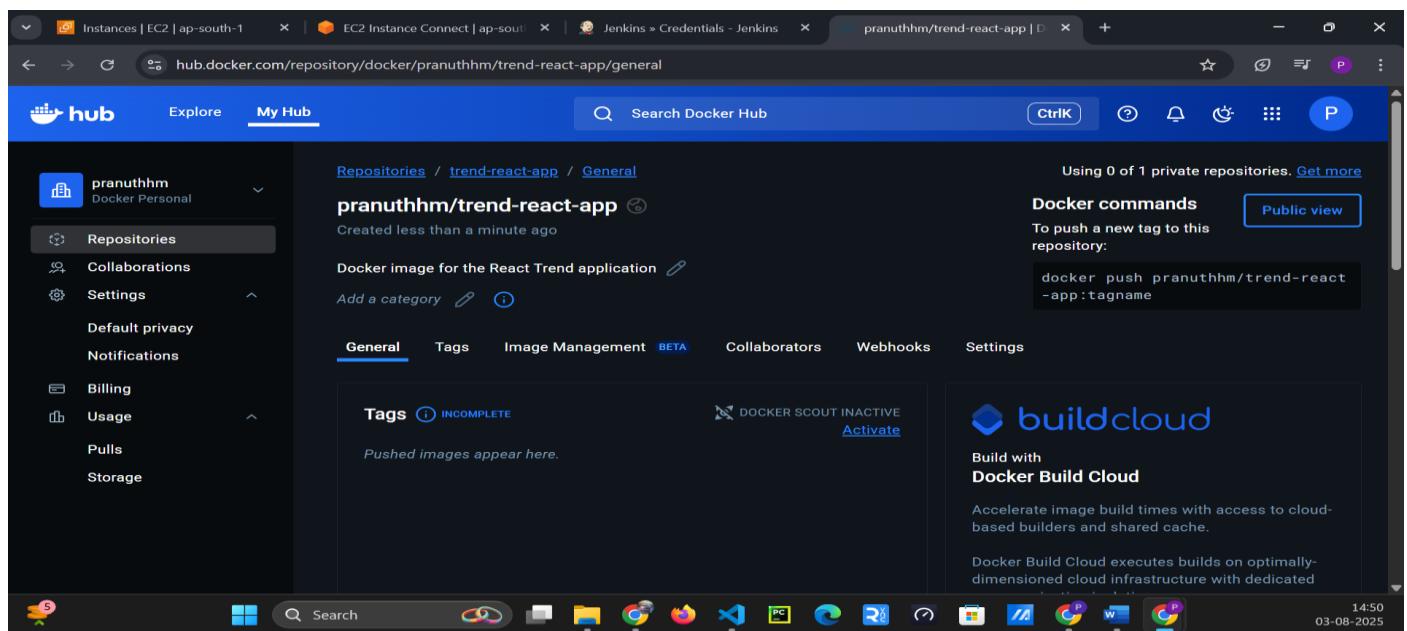
2. Create a New Repository:

- In the top navigation bar, click on **Repositories**.
- Click the **Create Repository** button.

3. Fill in the Repository Details:

- **Name:** This name needs to be consistent with what you will use in your Jenkins pipeline. A good naming convention is `your_dockerhub_username/application_name`.
 - Use the name: `your_dockerhub_username/trend-react-app`
 - Replace `your_dockerhub_username` with your actual DockerHub username.
- **Description:** You can add a short description, like Docker image for the React Trend application.
- **Visibility:** Set this to **Public**. This is important because it allows the Kubernetes cluster to pull the image without needing extra authentication credentials, simplifying the deployment process.

4. Click the **Create button.**



Step 6: Kubernetes Setup (AWS EKS)

Your Terraform user_data script has already installed awscli, eksctl, and kubectl for you. However, you need to ensure the IAM role of the Jenkins EC2 instance has the right permissions to create and manage EKS resources.

1. Verify IAM Role Permissions:

- Go to your AWS IAM console.
 - Go to Roles and find the role named ec2-jenkins-role.
 - Click on the role and check the Permissions tab.
 - Ensure the following policies are attached:
 - AmazonEKSClusterPolicy
 - AmazonEKSServicePolicy
 - AmazonEC2ContainerRegistryFullAccess (Even though we're using DockerHub, this is a standard permission for EKS node groups).

Create policy ->jsontab

{

"Version": "2012-10-17"

"Statement": /

{

"Effect": "Allow",

"Action": /

"eks:CreateCluster",

"eks:DescribeCluster",

"eks:UpdateClusterC

"eks:DeleteCluster",

"eks:TagResource",

"eks:UntagResource

"eks:ListTagsForReso

"eks:DescribeUpdate".

"eks:ListUpdates".

"eks:UpdateCluster

"eks>CreateNodegroup".

"eks::UpdateNodegroup()

```
"eks:DeleteNodegroup",
"eks:DescribeNodegroup",
"eks>ListNodegroups",
"eks:DescribeClusterVersions",
"iam:GetRole",
"iam>ListRoles",
"iam>CreateRole",
"iam:AttachRolePolicy",
"iam>DeleteRole",
"iam:DetachRolePolicy",
"iam:PassRole",
"ec2:DescribeKeyPairs",
"ec2>CreateTags",
"ec2:DescribeVpcs",
"ec2:DescribeSubnets",
"ec2:DescribeSecurityGroups",
"ec2>CreateSecurityGroup",
"ec2:RevokeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2>DeleteSecurityGroup",
"ec2>CreateNetworkInterface",
"ec2>DeleteNetworkInterface",
"ec2:DescribeNetworkInterfaces",
"ec2:AttachNetworkInterface",
"ec2:DetachNetworkInterface",
"ec2:RunInstances",
"ec2:DescribeInstances",
"ec2:TerminateInstances",
"ec2:DescribeImages",
"ec2:DescribeInstanceStatus",
"ec2:DescribeVolumes",
"ec2>CreateVolume",
"ec2:AttachVolume",
"ec2:DetachVolume",
```

```

    "ec2:DeleteVolume",
    "ecr:GetAuthorizationToken",
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage",
    "ssm:GetParameter"
  ],
  "Resource": "*"
}
]
}

```

- Add permissions -> Attach policies and add them.

6.2. Create EKS Cluster Configuration File (cluster.yaml)

This file will describe the EKS cluster you want to create. It's best practice to define it in a file rather than passing all parameters in the command line.

1. While you are logged into the Jenkins EC2 instance, create a new file named cluster.yaml in your home directory: `nano cluster.yaml`
2. Copy and paste the following content into the file. **You must replace the placeholders for the VPC and subnet IDs with the ones created by Terraform in Step 3.**

To find these IDs, go to your AWS Console:

- **VPC ID:** Go to VPC -> Your VPCs and find the one named Project-VPC. Copy its ID.
- **Subnet ID:** Go to VPC -> Subnets and find the one named Project-Public-Subnet. Copy its ID.

apiVersion: eksctl.io/v1alpha5

kind: ClusterConfig

metadata:

name: my-trend-cluster

region: ap-south-1

version: "1.29" # Use a stable Kubernetes version

```
vpc:  
  id: <YOUR_VPC_ID>  
  
  subnets:  
    public:  
      ap-south-1a:  
        id: <YOUR_PUBLIC_SUBNET_ID>  
  
      ap-south-1b:  
        id: <YOUR_PUBLIC_SUBNET_2_ID>  
  
  
  managedNodeGroups:  
    - name: my-trend-nodes  
      instanceType: t3.medium  
      desiredCapacity: 2  
      minSize: 1  
      maxSize: 3  
      volumeSize: 20 # GB  
      ssh:  
        allow: true  
        publicKeyPath: /home/ubuntu/.ssh/id_rsa.pub # We will create this key next
```

6.4. Create the EKS Cluster

Now, use eksctl with the configuration file to create the cluster.

1. Run the following command:

```
eksctl create cluster -f cluster.yaml+
```

6.5. Verify Cluster Creation and Configure kubectl

After eksctl finishes, you need to configure kubectl to communicate with the new cluster.

Before that configure AWS CLI in your instance

- i. sudo apt update && sudo apt upgrade -y

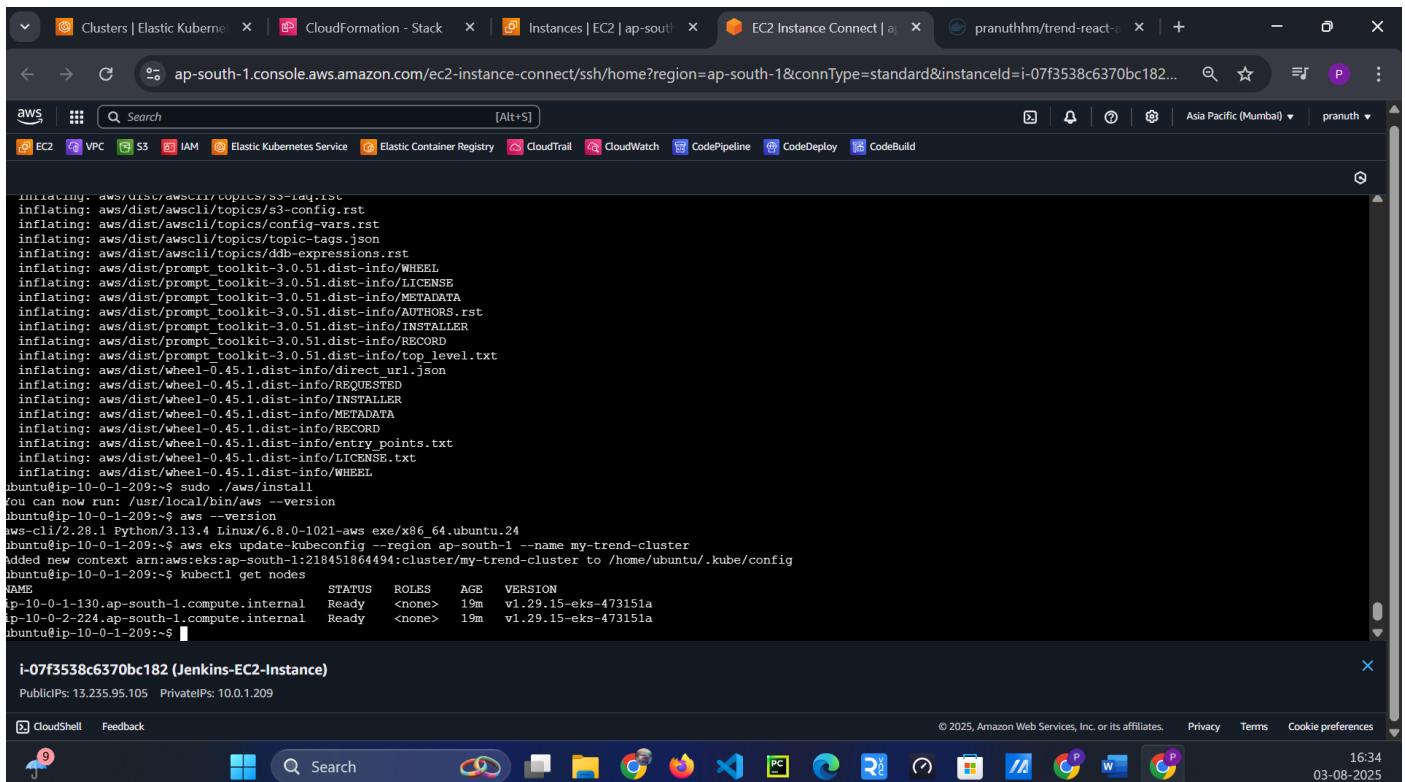
- ii. sudo apt install unzip curl -y
- iii. curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
- iv. unzip awscliv2.zip
- v. sudo ./aws/install
- vi. aws --version

1. Run the following command to update your kubeconfig file with the cluster information:

```
aws eks update-kubeconfig --region ap-south-1 --name my-trend-cluster
```

2. Now, verify that kubectl can see the cluster nodes:

```
kubectl get nodes
```



```

infating: aws/dist/awscli/topics/ss-ray.rst
infating: aws/dist/awscli/topics/s3-config.rst
infating: aws/dist/awscli/topics/config-vars.rst
infating: aws/dist/awscli/topics/topic-tags.json
infating: aws/dist/awscli/topics/dynamodb-expressions.rst
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/WHEEL
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/LICENSE
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/METADATA
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/NOTHORS.rst
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/INSTALLER
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/RECORD
infating: aws/dist/prompt_toolkit-3.0.51.dist-info/top_level.txt
infating: aws/dist/wheel-0.45.1.dist-info/direct_url.json
infating: aws/dist/wheel-0.45.1.dist-info/REQUESTED
infating: aws/dist/wheel-0.45.1.dist-info/INSTALLER
infating: aws/dist/wheel-0.45.1.dist-info/METADATA
infating: aws/dist/wheel-0.45.1.dist-info/RECORD
infating: aws/dist/wheel-0.45.1.dist-info/entry_points.txt
infating: aws/dist/wheel-0.45.1.dist-info/LICENSE.txt
infating: aws/dist/wheel-0.45.1.dist-info/WHEEL
ubuntu@ip-10-0-1-209:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
ubuntu@ip-10-0-1-209:~$ aws --version
aws-cli/2.28.1 Python/3.13.4 Linux/6.8.0-1021-aws exe/x86_64/ubuntu.24
ubuntu@ip-10-0-1-209:~$ aws eks update-kubeconfig --region ap-south-1 --name my-trend-cluster
added new context arn:aws:eks:ap-south-1:218451864494:cluster/my-trend-cluster to /home/ubuntu/.kube/config
ubuntu@ip-10-0-1-209:~$ kubectl get nodes
NAME                      STATUS    ROLES   AGE     VERSION
ip-10-0-1-130.ap-south-1.compute.internal  Ready   <none>   19m    v1.29.15-eks-473151a
ip-10-0-2-224.ap-south-1.compute.internal  Ready   <none>   19m    v1.29.15-eks-473151a
ubuntu@ip-10-0-1-209:~$ 
```

i-07f3538c6370bc182 (Jenkins-EC2-Instance) X

PublicIPs: 13.235.95.105 PrivateIPs: 10.0.1.209

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

16:34
03-08-2025

Step 7: Kubernetes Deployment Manifests

Now that you have a running Kubernetes cluster, you need to tell it how to deploy your application. You will do this by creating two YAML files: a Deployment to manage your application's pods and a Service to expose your application to the internet.

You will create these files on the Jenkins EC2 instance because your Jenkins pipeline will need to use them.

7.1. Create the deployment.yaml File: *nano deployment.yaml*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: trend-app-deployment
  labels:
    app: trend-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: trend-app
  template:
    metadata:
      labels:
        app: trend-app
    spec:
      containers:
        - name: trend-app-container
          image: pranuthhm/trend-react-app:latest
      ports:
        - containerPort: 3000
```

7.2. Create the service.yaml

This file creates a Service that provides a stable IP address and DNS name for your application. We will use the LoadBalancer type, which automatically provisions an AWS Network Load Balancer (NLB) to expose your application to the public internet.

```
apiVersion: v1
kind: Service
metadata:
  name: trend-app-service
  labels:
    app: trend-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
spec:
  selector:
    app: trend-app
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
```

Excellent! Step 6 is a major milestone, and it's great that your AWS EKS cluster is now successfully created and operational

7.3. Initial Deployment and Verification (Optional but Recommended)

It's a good practice to test these files manually before putting them in your Jenkins pipeline.

1. Deploy your application and service:

```
kubectl apply -f deployment.yaml
```

```
kubectl apply -f service.yaml
```

2. Verify the deployment:

```
kubectl get deployments
```

```
kubectl get pods
```

```
kubectl get services
```

The screenshot shows a CloudShell terminal window with several tabs at the top: Clusters | Elastic Kube... (active), CloudFormation - Stack, Instances | EC2 | ap-south..., EC2 Instance Connect | a..., and pranuthhm/trend-react-... (with a progress bar). The main area displays the output of various Kubernetes commands:

```
aws/distro/wheel-0.45.1.dist-info/entry_points.txt
inflating: aws/distro/wheel-0.45.1.dist-info/LICENSE.txt
inflating: aws/distro/wheel-0.45.1.dist-info/WHEEL
ubuntu@ip-10-0-1-209:~$ sudo ./aws/install
you can now run: /usr/local/bin/aws --version
ubuntu@ip-10-0-1-209:~$ aws --version
aws-cli/2.28.1 Python/3.13.4 Linux/6.8.0-1021-aws exe/x86_64/ubuntu.24
ubuntu@ip-10-0-1-209:~$ aws eks update-kubeconfig --region ap-south-1 --name my-trend-cluster
added new context arn:aws:eks:ap-south-1:21845186494:cluster/my-trend-cluster to /home/ubuntu/.kube/config
ubuntu@ip-10-0-1-209:~$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
p-10-0-1-130.ap-south-1.compute.internal Ready <none> 19m v1.29.15-eks-473151a
p-10-0-2-224.ap-south-1.compute.internal Ready <none> 19m v1.29.15-eks-473151a
ubuntu@ip-10-0-1-209:~$ nano deployment.yaml
ubuntu@ip-10-0-1-209:~$ kubectl apply -f deployment.yaml
deployment.apps/trend-app-deployment created
ubuntu@ip-10-0-1-209:~$ kubectl apply -f service.yaml
service/trend-app-service created
ubuntu@ip-10-0-1-209:~$ kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
trend-app-deployment 0/2 2 0 16s
ubuntu@ip-10-0-1-209:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
trend-app-deployment-5d966b4795-n5wqm 0/1 ImagePullBackOff 0 23s
trend-app-deployment-5d966b4795-wnwj5 0/1 ImagePullBackOff 0 23s
ubuntu@ip-10-0-1-209:~$ kubectl get services
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 172.20.0.1 <none> 443/TCP 31m
trend-app-service LoadBalancer 172.20.236.5 a03219eea78c24581bdfcafba159f248-1f1554729af94b0e.elb.ap-south-1.amazonaws.com 80:30900/TCP 43s
ubuntu@ip-10-0-1-209:~$
```

A modal window titled "i-07f3538c6370bc182 (Jenkins-EC2-Instance)" is open, showing Public IPs: 13.235.95.105 and Private IPs: 10.0.1.209.

7.4. Push to GitHub

The screenshot shows a GitHub repository page for "React_Application_Deployment_Document_2_Trend". The repository is public and has 8 commits. The commit history includes:

- Add Kubernetes deployment and service manifests
- Add .gitignore and .dockerignore files
- Add Dockerfile and Nginx config for application deployment
- Add Jenkinsfile for CI/CD pipeline
- Add Kubernetes deployment and service manifests
- Add Dockerfile and Nginx config for application deployment

The repository has 0 stars, 0 forks, and 0 watching. The "About" section describes the repository as a production-ready deployment of a React application utilizing Docker for containerization, Terraform for infrastructure as code, Kubernetes for orchestration, Jenkins for CI/CD automation, and monitoring with Prometheus and Grafana, all hosted on AWS Cloud.

Step 8: Jenkins CI/CD Pipeline: The Jenkinsfile is a text file that defines the steps of your pipeline. It lives in the root of your application's GitHub repository.

1. **On your local machine**, navigate to the root of your `Trend` project directory. This is the directory where you cloned the code and created the Docker files.
2. Create the Jenkinsfile
3. **Paste the following content.** This script has stages for building the Docker image, pushing it to DockerHub, and deploying it to Kubernetes. **Make sure to replace `pranuthhm/trend-react-app` with your actual DockerHub username and repository name.**

```
pipeline {  
  
    agent any  
  
    environment {  
  
        DOCKER_IMAGE = "pranuthhm/trend-react-app"  
  
        DOCKERHUB_CREDENTIALS = "dockerhub_credentials"  
  
        // Update the path to the new location  
  
        KUBECONFIG_PATH = "/var/lib/jenkins/.kube/config"  
  
    }  
  
    stages {  
  
        stage('Git Checkout') {  
  
            steps {  
  
                // This is handled automatically by the Git plugin  
  
                echo 'Checking out code from Git...'  
  
                checkout scm  
  
            }  
  
        }  
  
        stage('Build Docker Image') {  
  
            steps {  
  
                script {  
  
                    // Get the current Git commit hash for tagging the image  
  
                }  
  
            }  
  
        }  
  
    }  
  
}
```

```

def gitHash = sh(returnStdout: true, script: 'git rev-parse --short HEAD').trim()

// Build the Docker image with the Git commit hash as a tag

sh "docker build -t ${DOCKER_IMAGE}:${gitHash} -t ${DOCKER_IMAGE}:latest ."

}

}

}

stage('Push to DockerHub') {

steps {

withCredentials([usernamePassword(credentialsId: DOCKERHUB_CREDENTIALS,
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {

script {

// Login to DockerHub

sh "echo $DOCKER_PASS | docker login --username $DOCKER_USER --password-
stdin"

// Push the 'latest' and commit-tagged images to DockerHub

sh "docker push ${DOCKER_IMAGE}:latest"

sh "docker push ${DOCKER_IMAGE}:$(sh(returnStdout: true, script: 'git rev-parse --
short HEAD').trim())"

}

}

}

}

stage('Deploy to Kubernetes') {

steps {

sh """

# Configure kubectl to use the kubeconfig file on the Jenkins server

export KUBECONFIG=/var/lib/jenkins/.kube/config

# Apply the Kubernetes Deployment and Service manifests

```

```

echo "Applying Kubernetes Deployment...""

kubectl apply -f deployment.yaml

echo "Applying Kubernetes Service..."

kubectl apply -f service.yaml

"""

}

}

}

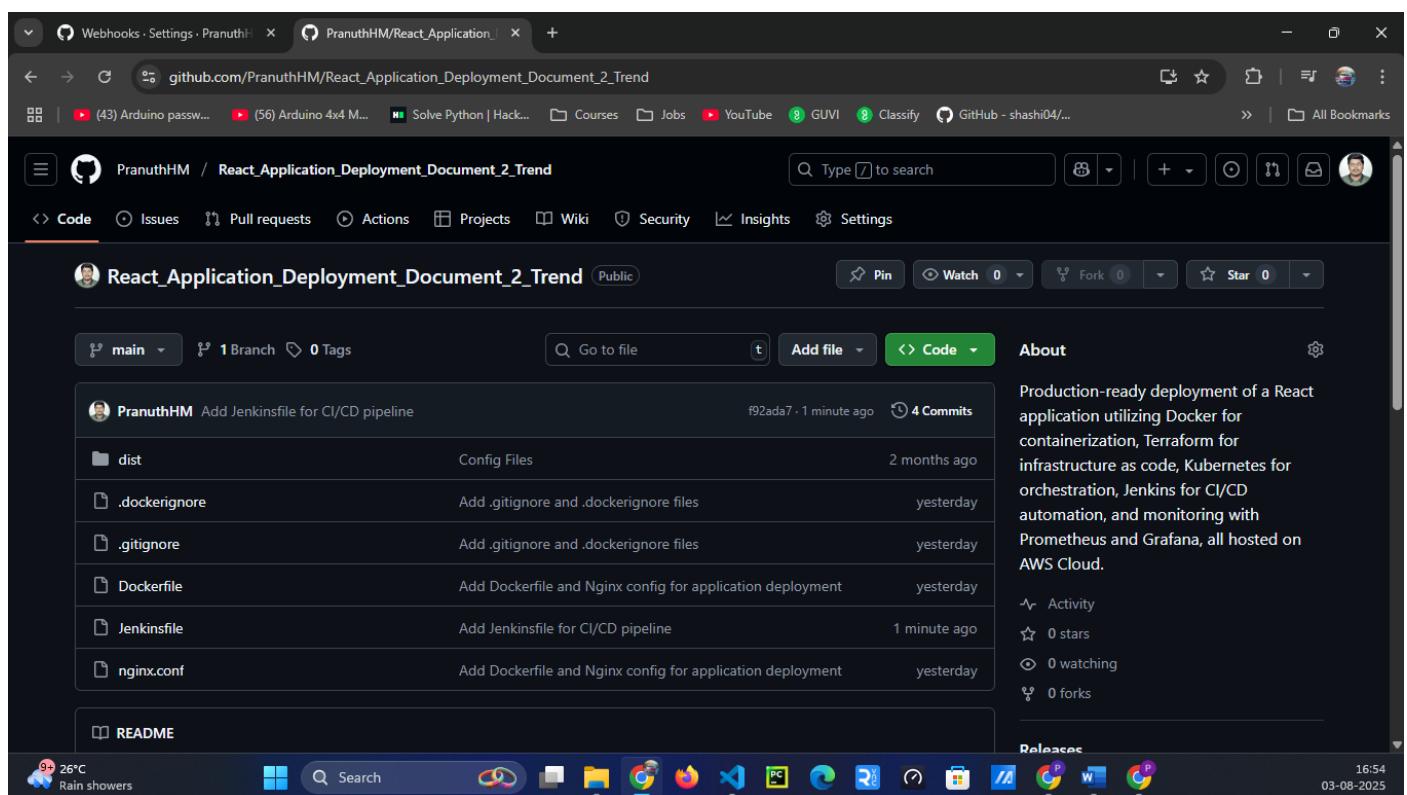
```

4. Commit and Push the Jenkinsfile:

```

git add Jenkinsfile
git commit -m "Add Jenkinsfile for CI/CD pipeline"
git push origin main

```



8.2. Create the Pipeline Job in Jenkins

8.2.1. Access your Jenkins UI: http://<YOUR_JENKINS_PUBLIC_IP>:8080

8.2.2. Create a New Item:

- Click New Item on the left-hand side.
- Enter an item name: Trend-App-CI-CD
- Select a type: Pipeline
- Click OK.

8.2.3. Configure the Pipeline:

- GitHub project: Enter your GitHub repository URL:
https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git
- Build Triggers: Check the GitHub hook trigger for GITScm polling box. This ensures a build is triggered on every Git push.
- Pipeline:
 - Definition: Select Pipeline script from SCM.
 - SCM: Select Git.
 - Repository URL: Enter your GitHub repository URL:
https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git
 - Credentials: Select - none - as your repository is public.
 - Branch Specifier: main
 - Script Path: Jenkinsfile

8.2.4. Save the job.

8.3. Trigger and Verify the Pipeline

- Go to your Jenkins job dashboard and click Build Now on the left.
- Wait for the build to complete. You can click on the build number to see the console output and watch each stage run.
- Once the pipeline completes successfully, go back to your SSH session on the Jenkins EC2 instance.
- Run kubectl get services and wait a few minutes for the EXTERNAL-IP (DNS name) of your service to be assigned. Copy the DNS name and paste it into your web browser. This time, you should see your application running!

Clusters | Elasti... | CloudFormat... | EC2 Instance | Trend-App-Cl... | pranuthm/tr... | pranuthm/tr... | aefc403705ad | +

Not secure 13.235.95.105:8080/job/Trend-App-Cl-CD/11/console

Jenkins / Trend-App-Cl-CD / #11

Status Changes Console Output

Download Copy View as plain text

Console Output

```

Started by user pranuth
Obtained Jenkinsfile from git https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Trend-App-Cl-CD
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Trend-App-Cl-CD/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git # timeout=10
Fetching upstream changes from https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git
> git --version # timeout=10
> git --version # 'git' version 2.43.0'
> git fetch --tags --force --progress -- https://github.com/PranuthHM/React_Application_Deployment_Document_2_Trend.git +refs/heads/*:refs/remotes/origin/*
timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 369672a43b9db278f9478897d528eaabbd18745 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 369672a43b9db278f9478897d528eaabbd18745 # timeout=10
Commit message: "Add Kubernetes deployment and service manifests"
> git rev-list --no-walk 1305ef55c1902db1c88ae08ca25f4e960c323fd7 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] finished with exit code 0

```

9+ Search

17:42 03-08-2025

Clusters | Elasti... | CloudFormat... | EC2 Instance | Trend-App-Cl... | pranuthm/tr... | pranuthm/tr... | Trendify | Find | +

ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=ap-south-1&connType=standard&instanceId=i-07f3538c6370bc182...

AWS Search [Alt+S]

EC2 VPC S3 IAM Elastic Kubernetes Service Elastic Container Registry CloudTrail CloudWatch CodePipeline CodeDeploy CodeBuild

```

ubuntu@ip-10-0-1-209:~$ kubectl get services
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP  172.20.0.1   <none>        443/TCP   52m
ubuntu@ip-10-0-1-209:~$ sudo chmod 755 /home/ubuntu/.kube
ubuntu@ip-10-0-1-209:~$ ls -la /home/ubuntu/.kube
total 20
drwxr-xr-x 3 ubuntu ubuntu 4096 Aug  3 11:03 .
drwxr-x--- 7 ubuntu ubuntu 4096 Aug  3 11:08 ..
drwxr-x--- 4 ubuntu ubuntu 4096 Aug  3 11:03 cache
-rw-r--r-- 1 ubuntu ubuntu 4619 Aug  3 11:03 config
ubuntu@ip-10-0-1-209:~$ sudo mkdir -p /var/lib/jenkins/.kube
ubuntu@ip-10-0-1-209:~$ sudo cp /home/ubuntu/.kube/config /var/lib/jenkins/.kube/config
ubuntu@ip-10-0-1-209:~$ sudo chown -R jenkins:jenkins /var/lib/jenkins/.kube
ubuntu@ip-10-0-1-209:~$ ls -la /var/lib/jenkins/.kube
total 16
drwxr-xr-x 2 jenkins jenkins 4096 Aug  3 11:56 .
drwxr-xr-x 19 jenkins jenkins 4096 Aug  3 11:56 ..
-rw-r--r-- 1 jenkins jenkins 4619 Aug  3 11:56 config
ubuntu@ip-10-0-1-209:~$ nano deployment.yaml
ubuntu@ip-10-0-1-209:~$ nano service.yaml
ubuntu@ip-10-0-1-209:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
trend-app-deployment-5d966b4795-llpf  1/1     Running   0          32s
trend-app-deployment-5d966b4795-kwzbv  1/1     Running   0          32s
ubuntu@ip-10-0-1-209:~$ kubectl get services
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP  172.20.0.1   <none>        443/TCP   90m
trend-app-service   LoadBalancer  172.20.155.98  aefc403705add4181a6eb5c271385318-e18b418990261ecb.elb.ap-south-1.amazonaws.com  80:31079/TCP   37s
ubuntu@ip-10-0-1-209:~$ ^C
ubuntu@ip-10-0-1-209:~$ 
```

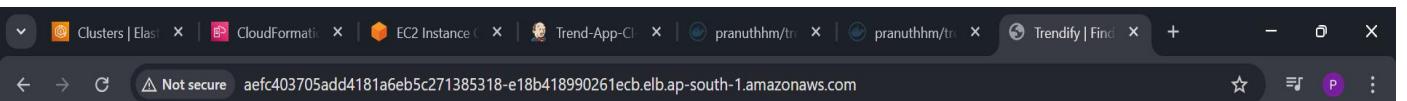
i-07f3538c6370bc182 (Jenkins-EC2-Instance)

Public IPs: 13.235.95.105 Private IPs: 10.0.1.209

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

17:50 03-08-2025



HOME COLLECTION ABOUT CONTACT



— OUR BEST SELLERS

Latest Arrivals

SHOP NOW —



Application deployed kubernetes Loadbalancer ARN:

<http://aefc403705add4181a6eb5c271385318-e18b418990261ecb.elb.ap-south-1.amazonaws.com/>

Step 9: Set up Prometheus and Grafana for Monitoring

9.1. Install Helm

First, you need to install Helm on your Jenkins EC2 instance if you haven't already.

1. SSH into your Jenkins EC2 instance.
2. Run the following commands to install Helm:

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

9.2. Install the Prometheus Stack with Helm

We'll use a Helm chart called `kube-prometheus-stack`, which is a comprehensive bundle that includes Prometheus, Grafana, and all the necessary components for Kubernetes monitoring.

1. Add the Prometheus Helm repository:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

```
helm repo update
```

2. Next, create a file named `grafana-values.yaml` on your Jenkins instance with the following content. This file tells Helm to create a public Load Balancer for Grafana.

```
# grafana-values.yaml
```

```
grafana:
```

```
  service:
```

```
    type: LoadBalancer
```

```
    port: 80
```

```
    targetPort: 3000
```

3. Install the `kube-prometheus-stack` chart. This will deploy Prometheus and Grafana to a new namespace called `monitoring`.

```
helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace
```

4. Verify that the pods are running in the `monitoring` namespace:

```
kubectl get pods -n monitoring
```

```

ubuntu@ip-10-0-1-209:~$ helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring -f grafana-values.yaml
NAME: prometheus
LAST DEPLOYED: Sun Aug 3 12:48:11 2022
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus"

Get Grafana 'admin' user password by running:
  kubectl --namespace monitoring get secrets prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo

Access Grafana local instance:
  export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=prometheus" -oname)
  kubectl --namespace monitoring port-forward $POD_NAME 3000
Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.

ubuntu@ip-10-0-1-209:~$ kubectl get services -n monitoring
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP          PORT(S)           AGE
alertmanager-operated   ClusterIP  None         <none>
prometheus-grafana     LoadBalancer 172.20.210.104  a526a2dc674941e58ae04aaeb501895-1503487297.ap-south-1.elb.amazonaws.com  9093/TCP,9094/TCP,9094/UDP  16s
prometheus-kube-prometheus-alertmanager   ClusterIP  172.20.224.87  <none>
prometheus-kube-prometheus-operator    ClusterIP  172.20.104.24  <none>
prometheus-kube-prometheus-prometheus  ClusterIP  172.20.166.35  <none>
prometheus-kube-state-metrics       ClusterIP  172.20.234.190 <none>
prometheus-operated             ClusterIP  None         <none>
prometheus-prometheus-node-exporter ClusterIP  172.20.206.65  <none>

ubuntu@ip-10-0-1-209:~$ [REDACTED]

```

i-07f3538c6370bc182 (Jenkins-EC2-Instance)
PublicIPs: 13.235.95.105 PrivateIPs: 10.0.1.209

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 18:26 03-08-2025

Look for the **prometheus-grafana** service. It will now have an AWS DNS name listed under the **EXTERNAL-IP** column. Copy this DNS name, paste it into your browser, and you should be able to see the Grafana login page.

Default Credentials:

- **Username:** admin
- **Password:** prom-operator

Welcome to Grafana

Need help? [Documentation](#) [Tutorials](#) [Community](#) [Public Slack](#)

Basic

The steps below will guide you to quickly finish setting up your Grafana installation.

TUTORIAL DATA SOURCE AND DASHBOARDS

Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

COMPLETE

Add your first data source

Remove this panel

Learn how in the docs ↗

Dashboards Latest from the blog

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 18:27 03-08-2025

Grafana

Clusters | Elasti... | Instance detail | EC2 Instance | Trend-App-Cl... | pranuthhm/tr... | Trendify | Find | Kubernetes /

Home > Dashboards > Kubernetes / Compute Resources / Pod

Search... ctrl+k

Data source default namespace default

Last 1 hour UTC Refresh 10s

pod trend-app-deployment-5d966b4795-l1p9f

Kubernetes

CPU Usage

0.00002
0.000015
0.00001
0.000005
12:00 12:05 12:10 12:15 12:20 12:25 12:30 12:35 12:40 12:45 12:50 12:55

Name Last *
trend-app-container 0.00000968

CPU Throttling

9+ Search

18:28 03-08-2025

This screenshot shows a Grafana dashboard for monitoring Kubernetes compute resources. The main panel displays a line chart titled 'CPU Usage' for a pod named 'trend-app-container'. The Y-axis represents CPU usage values from 0.000005 to 0.00002. The X-axis shows time from 12:00 to 12:55 UTC. A single green line shows the usage fluctuating between 0.00001 and 0.000015, with a significant spike reaching approximately 0.000018 at 12:50 UTC. Below the chart is a section titled 'CPU Throttling'.

Grafana

Clusters | Elasti... | Instance detail | EC2 Instance | Trend-App-Cl... | pranuthhm/tr... | Trendify | Find | Kubernetes /

Home > Dashboards > Kubernetes / Compute Resources / Clus...

Search... ctrl+k

Data source default

Last 1 hour UTC Refresh 10s

Kubernetes

CPU Utilisation 2.31% CPU Request... 18.1% CPU Limits C... No data Memory Utilis... 23.5% Memory Req... 11.2% Memory Limit... 17.2%

CPU Usage

0.07
0.06
0.05
0.04
0.03
0.02
0.01
0

Name Last *
default 0.0000221
kube-system 0.0139
monitoring 0.0219

CPU Quota

Namespace	Pods	Workloads	CPU Usage	CPU Requests	CPU Requests %
kube-system	8	4	0.0139	0.700	1.99%

9+ Search

18:29 03-08-2025

This screenshot shows a Grafana dashboard for monitoring Kubernetes cluster-level compute resources. It features several summary cards at the top displaying percentages: CPU Utilisation (2.31%), CPU Requests (18.1%), CPU Limits (No data), Memory Utilisation (23.5%), Memory Requests (11.2%), and Memory Limits (17.2%). Below these are two detailed charts. The first chart, 'CPU Usage', shows individual namespace usage over time, with three lines representing 'default', 'kube-system', and 'monitoring'. The second chart, 'CPU Quota', provides a breakdown of resource usage by namespace, showing 8 pods in the 'kube-system' namespace with a total CPU usage of 0.0139 and requests of 0.700, resulting in a 1.99% request percentage. The bottom navigation bar includes icons for various applications like Docker, Google Sheets, and Microsoft Word.

