Expert Cloud Consulting

Enhance Optimise & Scale

ASCP GPUonCLOUD Pvt Ltd

# “Expert Cloud Consulting” -

# SOP | CI/CD Pipeline with Jenkins

20.January.2025

version 1.0

—

Contributed by Tejal Kale
Approved by Akshay Shinde
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

# “Expert Cloud Consulting”

## CI/CD Pipeline with Jenkins

## 1.0 Contents


# 2.0 General Information:

## 2.1 Document Purpose

This document outlines the purpose, implementation details, and operational guidelines for our Jenkins-based CI/CD pipeline. The pipeline automates the process of code integration, testing, and deployment to ensure reliable and consistent software delivery.

## 2.2 Document Revisions

| Date | Version | Contributor(s) | Approver(s) | Section(s) | Change(s) |
|---|---|---|---|---|---|
| 20/Jan/2025 | 1.0 | Tejal Kale | Akshay Shinde | All Sections | New Document Created |
| | | | | | |

## 3.0 Document Overview:

This documentation covers:

- Setting up Jenkins CI/CD pipeline
- GitHub integration for automated code pulling
- Automated unit testing implementation
- Docker container build and deployment process
- Integration of Slack/email notifications for pipeline status alerts
- Best practices and troubleshooting guidelines

## 4.0 Project Overview:

### 4.1 Architecture

The CI/CD pipeline architecture consists of the following components:

1. Source Control:
   - GitHub repository for code hosting
2. CI/CD Server:
   - Jenkins server for pipeline orchestration
   - Pipeline stages: Build, Test, Deploy
   - Docker integration for containerization
3. Testing Environment:
   - Automated unit test execution
   - Test result reporting and analysis
4. Deployment:
   - Staging environment setup
   - Docker container registry
   - Automated deployment processes
5. Notification System:
   - Email notification service
   - Alert configuration for pipeline events

### 4.2 Prerequisites:

**Software Requirements:**

- Jenkins (version 2.400.x or higher)
- Git (version 2.34.x or higher)
- Docker (version 24.x or higher)
- Java Development Kit (JDK 17 or higher)

**Access Requirements:**

- GitHub repository access
- Jenkins admin credentials
- Docker registry credentials
- Email app password (for notifications)

## 5.0 Install and Set Up Jenkins:

### 5.1 Install Jenkins:

1. Update your system:
   # sudo apt update && sudo apt upgrade -y

2. Installation of Java :
   Jenkins requires Java to run, yet not all Linux distributions include Java by default.
   # sudo apt install fontconfig openjdk-17-jre
   java -version

```
ubuntu@ip-172-31-4-55:~$ java -version
openjdk version "17.0.13" 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Ubuntu-2ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Ubuntu-2ubuntu124.04, mixed mode, sharing)
```

3. Add the Jenkins repository and install Jenkins:

   # sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \ https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

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

   # sudo apt-get install Jenkins

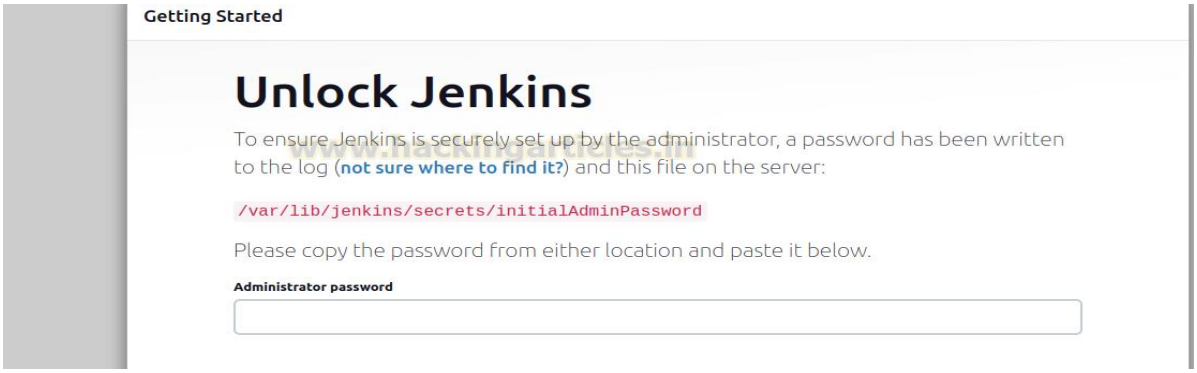
4. You can enable the Jenkins service to start at boot with the command:
   # sudo systemctl enable Jenkins

5. You can start the Jenkins service with the command:
   # sudo systemctl start Jenkins

6. You can check the status of the Jenkins service using the command:
   # sudo systemctl status Jenkins

```
ubuntu@ip-172-31-4-55:~$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-4-55:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-4-55:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-01-14 06:09:32 UTC; 51s ago
   Main PID: 4048 (java)
      Tasks: 41 (limit: 1130)
     Memory: 313.8M (peak: 353.7M)
        CPU: 15.027s
     CGroup: /system.slice/jenkins.service
             └─4048 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war
```

7. If everything has been set up correctly, you should see an output like this:

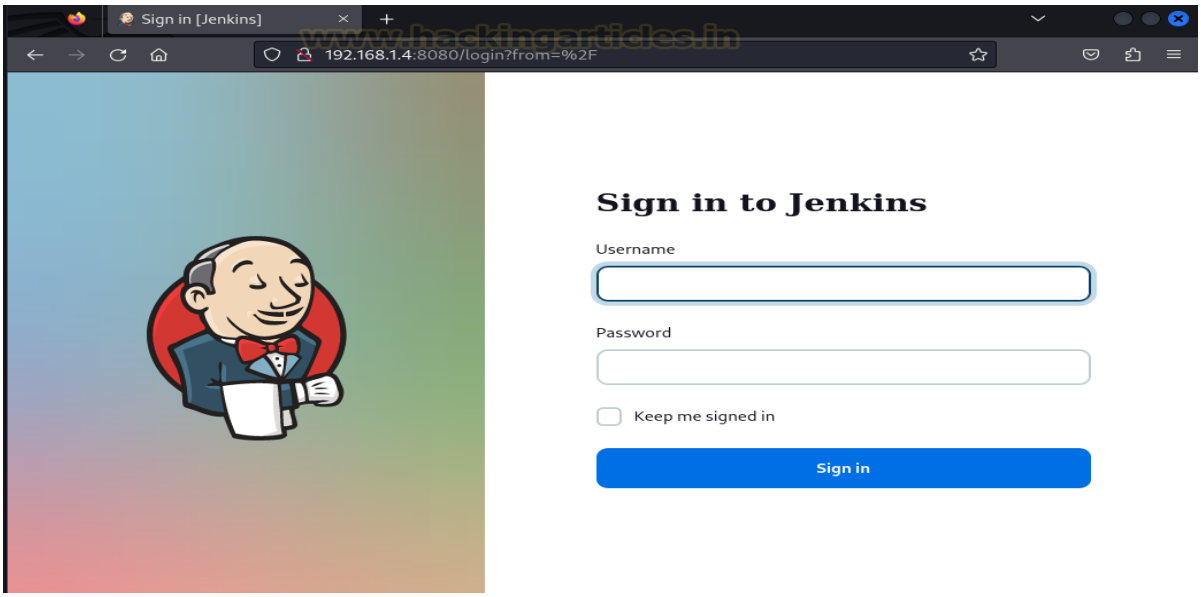



8. Password can be obtained by reading the content of the **initialAdminPassword** file.

   # cat /var/lib/Jenkins/secrets/initialAdminPassword

9. Select the **Install suggested plugins** to **Customize Jenkins** and proceed with the installation. After successfully installing and configuring the Jenkins server, we can start the exploitation using the machine. Starting with the enumeration, since at port 8080 the Jenkins Server is running in the ubuntu machine hence checking the port 8080.

   Access Jenkins on http://<your-server-ip>:8080




### 5.2 Install Required Plugins

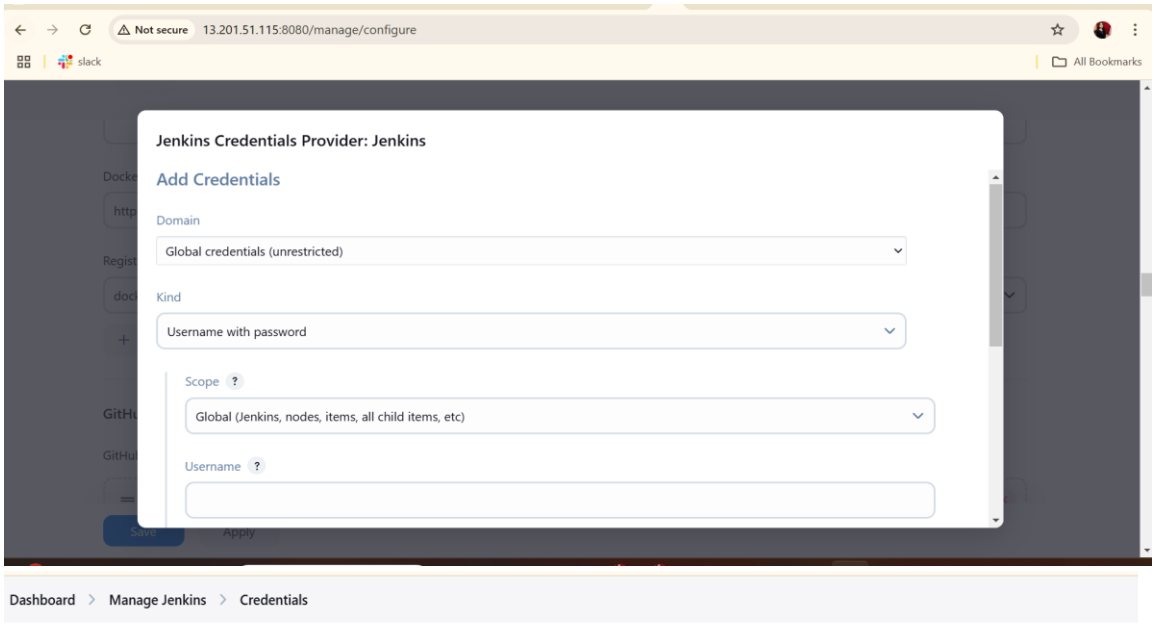
Install the following Jenkins plugins:

- Git
- Pipeline
- Email

- Docker

### 5.3 Configure Jenkins Credentials

1. Add GitHub credentials in Jenkins:
   - Go to Manage Jenkins > Credentials.
   - Add a new credential with your GitHub username and personal access token.
2. Add Docker Hub credentials for container deployment if necessary.




Credentials

| T | P | Store ↓ | Domain | ID | Name |
|---|---|---|---|---|---|
| | | System | (global) | cicd-jenkins-github | cicd-jenkins-github |
| | | System | (global) | docker_login | docker_login/****** |
| | | System | (global) | docker_login_new | docker_login_new/****** |
| | | System | (global) | tejalkale13 | tejalkale13 |
| | | System | (global) | docker | tejalkale13/****** |
| | | System | (global) | gmail | tejalbkale@gmail.com/****** |

## 6.0 Create a GitHub Repository

1. Push your project code to a GitHub repository.
2. Ensure the repository contains:
   - Dockerfile
   - Unit test scripts
   - Jenkinsfile

## 7.0 Create a Jenkinsfile

A Jenkinsfile is a script that tells Jenkins what steps to take in a pipeline. It organizes tasks like pulling code, running tests, building, and deploying your app.

### 7.1 Set Up Environment

**Purpose**: Define environment variables to ensure consistency across stages.

**Steps**:

- Use the environment directive to declare variables such as image names or repository paths.
- These variables can be reused in subsequent stages.

Script ?

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = "sample-nodejs-app"
        NODE_ENV = "staging"
        DOCKER_CREDENTIALS = "docker"
        DOCKER_USERNAME = "tejalkale13"
        DOCKER_TAG = "1.0.0"
    }
```

### 7.2 Clone Repository

**Purpose**: Fetch the latest code from the GitHub repository.

**Steps**:

- Use the Git plugin in Jenkins.
- Provide the repository URL and credentials.

```
    stage('Checkout') {
        steps {
            script {
                echo "Cloning the repository..."
                git url: 'https://github.com/Tejalkale13/sample-nodejs-app.git',
                branch: 'main',
                credentialsId: 'cicd-jenkins-github'
            }
        }
    }
```

### 7.3 Install Dependencies

**Purpose**: Install required dependencies to build and run the application.

**Steps**:

- Use a package manager (e.g., npm or pip) to install dependencies listed in the project.
- Ensure dependencies are installed before running tests or building the application.

Script ?

```
    stage('Install Dependencies') {
        steps {
            script {
                echo "Installing Node.js dependencies..."
                sh 'npm cache clean --force && npm install'
            }
        }
    }
```

### 7.4 Run Unit Tests

**Purpose**: Automatically test the code to ensure it works as expected.

**Steps**:

- Run test scripts using tools like unit test.

- Capture test results in Jenkins.

```
29      stage('Test') {
30          steps {
31              echo "Running the tests..."
32              sh 'fuser -k 3000/tcp || true && npm test'
33          }
34      }
```

### 7.5 Build Docker Image

**Purpose**: Package the application into a Docker container.

**Steps**:

- Use the docker build command.
- Specify the Dockerfile and tag the image.

```
35      stage('Build Docker Image') {
36          steps {
37              script {
38                  echo "Building Docker image..."
39                  sh "docker build -t ${DOCKER_IMAGE}:${DOCKER_TAG} . --no-cache"
40              }
41          }
42      }
```

### 7.6 Push Docker Image to Repository

**Purpose**: Upload the Docker image to Docker Hub or another container registry.

**Steps**:

- Tag the Docker image.
- Push the image using Docker credentials.

```
43  stage("Push to Docker Hub") {
44      steps {
45          echo "Pushing the image to Docker Hub..."
46  withCredentials([usernamePassword(credentialsId: 'docker', usernameVariable: 'USERNAME', passwordVariable: 'PA
47              script {
48                  // Login to Docker Hub
49                  sh "echo \$PASSWORD | docker login --username \$USERNAME --password-stdin"
50
51                  // Tag the image
52                  sh "docker tag ${DOCKER_IMAGE}:${DOCKER_TAG} ${DOCKER_USERNAME}/${DOCKER_IMAGE}:${DOCKER_TAG}"
53                  sh "docker tag ${DOCKER_IMAGE}:${DOCKER_TAG} ${DOCKER_USERNAME}/${DOCKER_IMAGE}:latest"
54
55                  // Push the images
56                  sh "docker push ${DOCKER_USERNAME}/${DOCKER_IMAGE}:${DOCKER_TAG}"
57                  sh "docker push ${DOCKER_USERNAME}/${DOCKER_IMAGE}:latest"
58
59                  // Cleanup
60                  sh "docker logout"
61              }
62          }
63      }
64  }
65
```

### 7.7 Deploy to Staging Environment

**Purpose**: Run the Docker container in a staging environment for testing.

**Steps**:

- Use docker run to start the container.
- Expose necessary ports.

```
    stage('Deploy') {
        steps {
            script {
                echo "Deploying to staging environment..."
                sh """
                    docker run -d \
                        --name nodejs-staging \
                        -p 3000:3000 \
                        -e NODE_ENV=${NODE_ENV} \
                        ${DOCKER_USERNAME}/${DOCKER_IMAGE}:latest
                """
            }
        }
    }
}
```

### 7.8 Notifications and Post Actions

**Purpose**: Inform the team of pipeline status.

**Steps**:

- Send email notifications for failures.

Script ?

```
}
post {
    success {
        echo "Pipeline completed successfully!"
    }
    failure {
        echo "Pipeline failed!"
        // Cleanup Docker containers
        sh '''
            docker ps -q --filter 'name=todo-app' | xargs -r docker stop
            docker ps -aq --filter 'name=todo-app' | xargs -r docker rm
        '''
        // Send failure email notification
        emailext (
            to: 'tejalbkale@gmail.com',
            subject: "Build failed: ${env.JOB_NAME} - Build # ${env.BUILD_NUMBER}",
            body: """
            The build for ${env.JOB_NAME} - Build # ${env.BUILD_NUMBER} has failed.
            Check the build logs for more details: ${env.BUILD_URL}
            """
        )
    }
}
}
```

### 7.9 Integrating Notifications

Configure Email Notifications

- Install the "Email Extension" plugin.
- Set up email configuration:
- Go to Manage Jenkins > Configure System.
- Scroll to "Extended Email Notification" and provide SMTP server details.
- Add emailext steps to your Jenkinsfile.

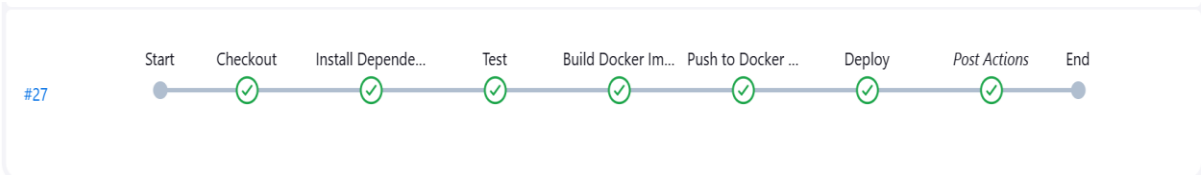
## 8.0 Testing Steps

**Purpose**: Ensure the pipeline builds successfully and notifications are sent correctly.

### 8.1 Validate Pipeline Build

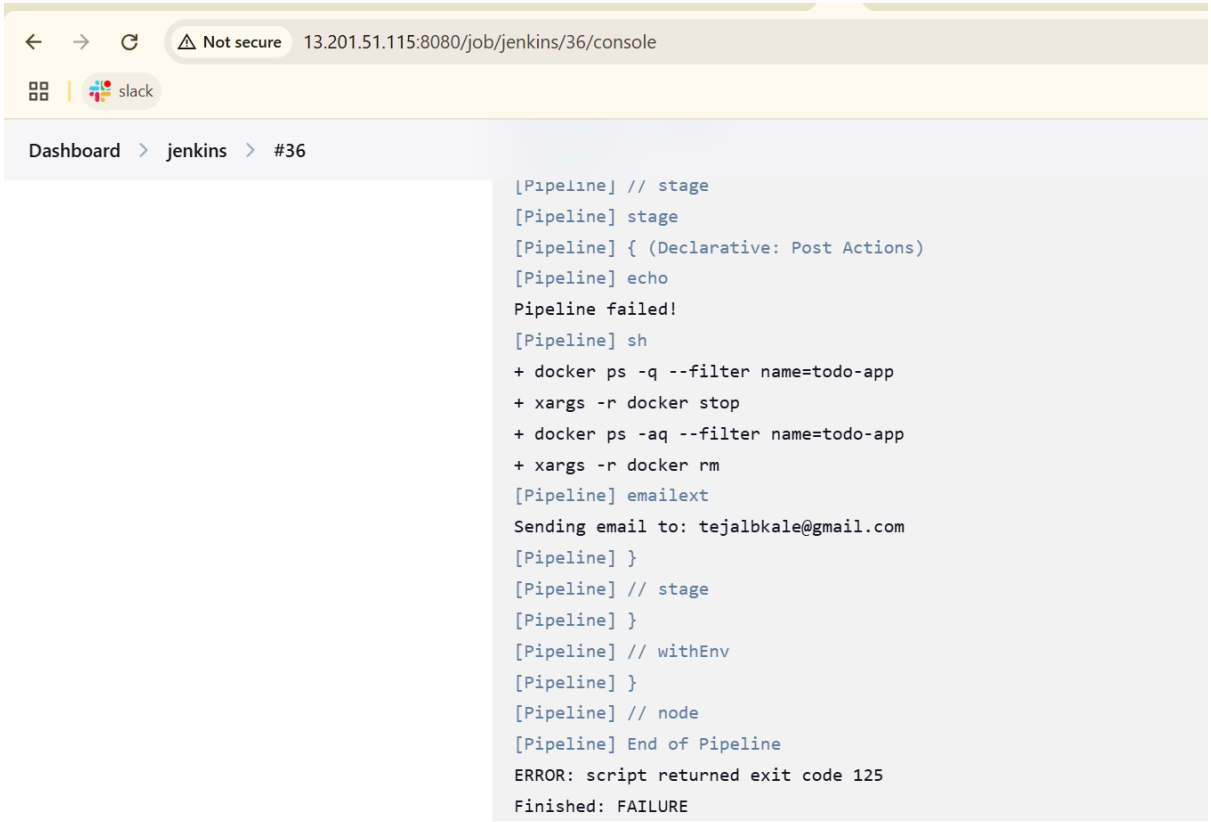
- Trigger the pipeline by pushing changes to the GitHub repository.
- Verify that all stages (Clone, Environment Setup, Dependencies, Testing, etc.) execute without errors.

- Check the logs in Jenkins for successful completion of each stage.



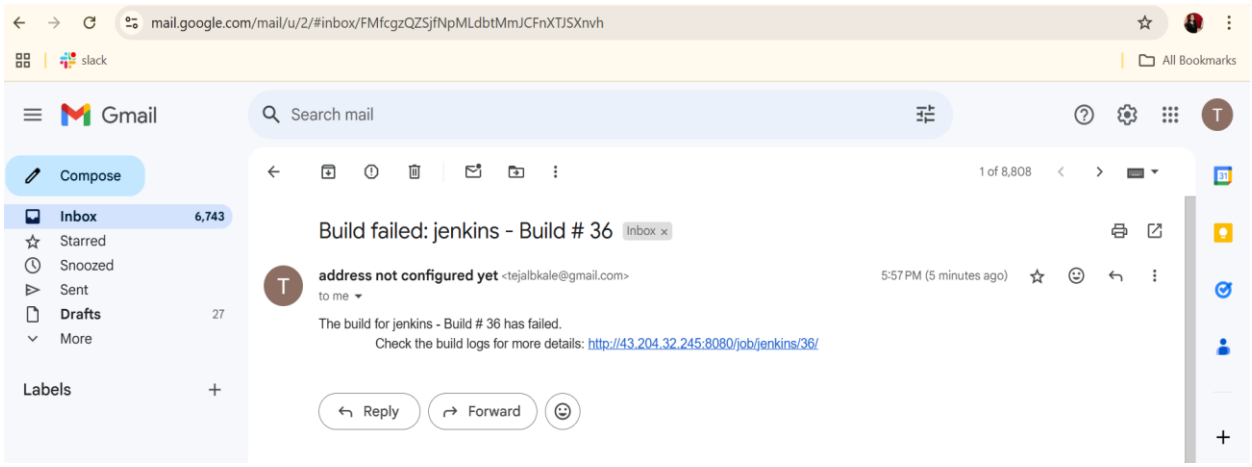

### 8.2 Test Email Notifications

1. Simulate a failure by introducing an error in the repository (e.g., a syntax error in code).
2. Observe the pipeline failure.
3. Verify that email notifications are sent to the configured recipients with error details.




### 8.3 Test Email Notifications

Verifying that email notifications are sent to the configured recipients with error details.


mail.google.com/mail/u/2/#inbox/FMfcgzQZSjfNpMLdbtMmJCFnXTJSXnvh
slack
All Bookmarks
Gmail
Search mail
Compose
Inbox 6,743
Starred
Snoozed
Sent
Drafts 27
More
Labels
1 of 8,808
Build failed: jenkins - Build # 36
Inbox ×
address not configured yet <tejalbkale@gmail.com>
to me
5:57 PM (5 minutes ago)
The build for jenkins - Build # 36 has failed.
Check the build logs for more details: http://43.204.32.245:8080/job/jenkins/36/
Reply
Forward