@author:JuanTous

CI/CD

ISS Communication System - NASA

V1 R

Overview

The ISS Communication System - NASA requires a robust Continuous Integration and Continuous Deployment (CI/CD) pipeline to ensure the reliability and efficiency of the system. This plan outlines a comprehensive approach to implementing CI/CD using Jenkins and Docker, based on official documentation and best practices.

Tools and Technologies

1. Jenkins
2. Docker
3. GitHub

CI/CD Pipeline

Stage 1: Source Code Management

1. GitHub: Store the ISS Communication System code in a GitHub repository.
2. Jenkins: Configure Jenkins to poll the GitHub repository for changes.

Stage 2: Build and Package

1. Jenkins: Use Jenkins to build the ISS Communication System code using a suitable build tool (e.g., Maven or Gradle).
2. Docker: Create a Docker image for the built application using a Dockerfile.
3. Jenkins: Push the Docker image to a container registry (e.g., Docker Hub).

Stage 3: Testing

1. Jenkins: Run automated tests (e.g., unit tests, integration tests) using a testing framework (e.g., JUnit, PyUnit).
2. Jenkins: Report test results and store them in a database (e.g., Jenkins' built-in database).

Stage 4: Deployment

1. Jenkins: Deploy the Docker image to a target environment (e.g., a cloud platform, a virtual machine).
2. Jenkins: Configure the deployment environment to use the deployed Docker image.

Stage 5: Monitoring and Feedback

1. Jenkins: Monitor the deployed application for errors, performance issues, and other metrics.
2. Jenkins: Provide feedback to developers through email, chat, or other notification channels.

Jenkins Configuration

1. Install Jenkins: Install Jenkins on a suitable server or cloud platform.
2. Configure Jenkins: Configure Jenkins to use the GitHub repository, Docker, and other required tools.
3. Create Jobs: Create Jenkins jobs for each stage of the pipeline (build, test, deploy, monitor).

Docker Configuration

1. Create a Dockerfile: Create a Dockerfile for the ISS Communication System application.
2. Build a Docker Image: Build a Docker image using the Dockerfile.
3. Push to Container Registry: Push the Docker image to a container registry (e.g., Docker Hub).

SCaN Testbed CI/CD

AUTOMATED TESTING

Pytest and Unittest to ensure that the ISS Communication System software meets the required standards and specifications

This test suite includes:

1. **`TestISSCommSystem`:** A unittest test case class that tests the basic functionality of the ISS Communication System module.
   - `test_init`: Verifies that the ISS Communication System module is initialized correctly.
   - `test_send_message`: Verifies that the `send_message` method sends a message correctly.
   - `test_receive_message`: Verifies that the `receive_message` method receives a message correctly.
   - `test_connect_to_iss`: Verifies that the `connect_to_iss` method connects to the ISS correctly.
   - `test_disconnect_from_iss`: Verifies that the `disconnect_from_iss` method disconnects from the ISS correctly.
2. **`TestISSCommSystemErrorHandling`:** A unittest test case class that tests the error handling of the ISS Communication System module.
   - `test_send_message_error`: Verifies that the `send_message` method raises an exception when an error occurs.
   - `test_receive_message_error`: Verifies that the `receive_message` method raises an exception when an error occurs.
   - `test_connect_to_iss_error`: Verifies that the `connect_to_iss` method raises an exception when an error occurs.
   - `test_disconnect_from_iss_error`: Verifies that the `disconnect_from_iss` method raises an exception when an error occurs.
3. **`test_validate_message`:** A Pytest test function that tests the `validate_message
4. `test_validate_iss_comm_system_config`: A Pytest test function that tests the `validate_iss_comm_system_config` method.
5. * Verifies that the method returns `True` for a valid ISS Communication System configuration and `False` for an invalid configuration.

```
# tests/test_iss_comm_system.py

import pytest
import unittest
from unittest.mock import patch, MagicMock
from iss_comm_system import ISSCommSystem  # Import the ISS
Communication System module

class TestISSCommSystem(unittest.TestCase):
    def setUp(self):
        self.iss_comm_system = ISSCommSystem()

    def test_init(self):
        self.assertIsInstance(self.iss_comm_system, ISSCommSystem)

    def test_send_message(self):
        message = "Hello, World!"
        self.iss_comm_system.send_message(message)
        self.assertEqual(self.iss_comm_system.sent_messages,
[message])

    def test_receive_message(self):
        message = "Hello, World!"
        self.iss_comm_system.receive_message(message)
        self.assertEqual(self.iss_comm_system.received_messages,
[message])

    def test_connect_to_iss(self):
        with patch('iss_comm_system.ISSCommSystem.connect_to_iss') as
mock_connect:
            self.iss_comm_system.connect_to_iss()
            mock_connect.assert_called_once()

    def test_disconnect_from_iss(self):
        with
patch('iss_comm_system.ISSCommSystem.disconnect_from_iss') as
mock_disconnect:
```

```
            self.iss_comm_system.disconnect_from_iss()
            mock_disconnect.assert_called_once()

class TestISSCommSystemErrorHandling(unittest.TestCase):
    def setUp(self):
        self.iss_comm_system = ISSCommSystem()

    def test_send_message_error(self):
        with patch('iss_comm_system.ISSCommSystem.send_message') as
mock_send:
            mock_send.side_effect = Exception("Error sending
message")
            with self.assertRaises(Exception):
                self.iss_comm_system.send_message("Hello, World!")

    def test_receive_message_error(self):
        with patch('iss_comm_system.ISSCommSystem.receive_message')
as mock_receive:
            mock_receive.side_effect = Exception("Error receiving
message")
            with self.assertRaises(Exception):
                self.iss_comm_system.receive_message("Hello, World!")

    def test_connect_to_iss_error(self):
        with patch('iss_comm_system.ISSCommSystem.connect_to_iss') as
mock_connect:
            mock_connect.side_effect = Exception("Error connecting to
ISS")
            with self.assertRaises(Exception):
                self.iss_comm_system.connect_to_iss()

    def test_disconnect_from_iss_error(self):
        with
patch('iss_comm_system.ISSCommSystem.disconnect_from_iss') as
mock_disconnect:
            mock_disconnect.side_effect = Exception("Error
disconnecting from ISS")
            with self.assertRaises(Exception):
                self.iss_comm_system.disconnect_from_iss()
```

```
@pytest.mark.parametrize("message, expected_result", [
    ("Hello, World!", True),
    ("", False),
    (None, False),
])
def test_validate_message(message, expected_result):
    iss_comm_system = ISSCommSystem()
    result = iss_comm_system.validate_message(message)
    assert result == expected_result

@pytest.mark.parametrize("iss_comm_system_config, expected_result", [
    ({'username': 'nasa', 'password': 'password'}, True),
    ({'username': '', 'password': 'password'}, False),
    ({'username': 'nasa', 'password': ''}, False),
])
def test_validate_iss_comm_system_config(iss_comm_system_config,
expected_result):
    iss_comm_system = ISSCommSystem()
    result =
iss_comm_system.validate_iss_comm_system_config(iss_comm_system_confi
g)
    assert result == expected_result
```

This will execute the tests and report any failures or errors.

Note that this is just an example test suite, and you may need to add or modify tests to cover specific requirements or edge cases for the ISS Communication System software.

Here is an example of how you could use Pytest fixtures to set up and tear down the ISS

Communication System module:
`

```
``python
# tests/conftest.py

import pytest
from iss_comm_system import ISSCommSystem

@pytest.fixture
def iss_comm_system():
    return ISSCommSystem()

@pytest.fixture
def mock_iss_comm_system():
    with patch('iss_comm_system.ISSCommSystem') as mock_iss_comm_system:
        yield mock_iss_comm_system
```

This defines two fixtures: `iss_comm_system` and `mock_iss_comm_system`. The `iss_comm_system` fixture creates a new instance of the ISS Communication System module, while the `mock_iss_comm_system` fixture creates a mock object for the ISS Communication System module using the `patch` function from `unittest.mock`

Jenkins Configuration

4. Install Jenkins: I did a bash file to automate this process called Jenkins on a suitable server or cloud platform.

```
#!/bin/bash

# Set the Jenkins version to install
JENKINS_VERSION="2.303"

# Set the Java version to install
JAVA_VERSION="1.8.0_242"
```

```
# Set the repository URL for Jenkins
JENKINS_REPO_URL="https://pkg.jenkins.io/debian-stable"

# Set the repository URL for Java
JAVA_REPO_URL="https://adoptopenjdk.jfrog.io/adoptopenjdk/rpm/centos/
7/x86_64"

# Install Java
echo "Installing Java $JAVA_VERSION..."
sudo yum install -y $JAVA_REPO_URL/$JAVA_VERSION

# Set the Java home environment variable
export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk

# Install Jenkins
echo "Installing Jenkins $JENKINS_VERSION..."
sudo wget -q -O - $JENKINS_REPO_URL/jenkins.io.key | sudo apt-key add
-
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install -y jenkins

# Configure Jenkins to use the correct Java version
echo "Configuring Jenkins to use Java $JAVA_VERSION..."
sudo update-alternatives --install /usr/bin/java java
/usr/lib/jvm/jre-1.8.0-openjdk/bin/java 1
sudo update-alternatives --set java
/usr/lib/jvm/jre-1.8.0-openjdk/bin/java

# Start Jenkins
echo "Starting Jenkins..."
sudo systemctl start jenkins

# Enable Jenkins to start at boot
echo "Enabling Jenkins to start at boot..."
sudo systemctl enable jenkins
```

```
# Configure the Jenkins URL
echo "Configuring Jenkins URL..."
sudo /usr/sbin/jenkins config set --httpPort=8080

# Restart Jenkins
echo "Restarting Jenkins..."
sudo systemctl restart jenkins

# Print the Jenkins URL
echo "Jenkins is installed and available at http://localhost:8080"
```

Plugins

- Workflow Aggregator Plugin
- Build Pipeline Plugin
- Multijob Plugin
- Git Plugin
- GitHub Integration Plugin
- Performance Plugin
- Mailer
- JUnit
- Performance Publisher Plugin

Here a bash file to automated this task and configures Jenkins, and then installs the specified plugins:

```
#!/bin/bash

# Install Jenkins
echo "Installing Jenkins..."
sudo wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key
| sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install -y jenkins
```

```
# Start Jenkins
echo "Starting Jenkins..."
sudo systemctl start jenkins

# Enable Jenkins to start at boot
echo "Enabling Jenkins to start at boot..."
sudo systemctl enable jenkins

# Wait for Jenkins to start
echo "Waiting for Jenkins to start..."
sleep 30

# Install plugins
echo "Installing plugins..."
sudo /usr/local/bin/jenkins-cli install-plugin workflow-aggregator
sudo /usr/local/bin/jenkins-cli install-plugin build-pipeline
sudo /usr/local/bin/jenkins-cli install-plugin multijob
sudo /usr/local/bin/jenkins-cli install-plugin git
sudo /usr/local/bin/jenkins-cli install-plugin github
sudo /usr/local/bin/jenkins-cli install-plugin performance
sudo /usr/local/bin/jenkins-cli install-plugin mailer
sudo /usr/local/bin/jenkins-cli install-plugin junit
sudo /usr/local/bin/jenkins-cli install-plugin performance-publisher

# Restart Jenkins
echo "Restarting Jenkins..."
sudo systemctl restart jenkins

echo "Jenkins is installed and configured with the specified
plugins."
```

Jenkinsfile (Groovy script)

```
pipeline {
    agent any

    stages {
      stage('Build') {
            steps {
                git
'https://gitlab.com/nasa/iss-communication-system.git'
                sh 'mvn clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('Deploy') {
            steps {
                sh 'mvn deploy'
            }
        }
    }
    post {
        always {
            mail to: 'nasa-dev-team@example.com',
                  subject: 'ISS Communication System Build Result',
                  body: 'Build result: ${currentBuild.result}'
        }
    }
}
```

CD

DOCKER

NASA's Public Docker Images:

1. nasa/cumulus: This is a Docker image for the Cumulus cloud-based data processing and analysis platform, which is used by NASA's Earth Science Division.
2. nasa/earthdata: This Docker image provides a set of tools and libraries for working with Earth science data, including data from NASA's Earth Observing System (EOS).
3. nasa/helioviewer: This Docker image provides a web-based interface for visualizing and exploring heliophysics data from NASA's Solar Dynamics Observatory (SDO) and other sources.

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'docker build -t iss-comm-system .'
            }
        }
        stage('Test') {
            steps {
                sh 'docker run -t iss-comm-system'
            }
        }
        stage('Deploy') {
            steps {
                sh 'docker tag iss-comm-system:latest
iss-comm-system:prod'
                sh 'docker push iss-comm-system:prod'
                sh 'kubectl apply -f deployment.yaml'
            }
        }
    }
}
```

Here is an example of a **Jenkinsfile** that demonstrates a basic CI/CD pipeline for the ISS Communication System:

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'docker build -t iss-comm-system .'
            }
        }
        stage('Test') {
            steps {
                sh 'docker run -t iss-comm-system'
            }
        }
        stage('Deploy') {
            steps {
                sh 'docker tag iss-comm-system:latest
iss-comm-system:prod'
                sh 'docker push iss-comm-system:prod'
                sh 'kubectl apply -f deployment.yaml'
            }
        }
    }
}
```

Deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iss-comm-system
  labels:
    app: iss-comm-system
spec:
  replicas: 2
  selector:
    matchLabels:
      app: iss-comm-system
  template:
    metadata:
      labels:
        app: iss-comm-system
    spec:
      containers:
      - name: iss-comm-system
        image: nasa/iss-comm-system:latest
        ports:
        - containerPort: 8080
```

CD

Continuous Deployment (CD) pipeline for deploying the ISS Communication System software to the International Space Station (ISS):

Jenkins job file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <actions/>
  <description>Build and deploy NASA project</description>
  <keepDependencies>false</keepDependencies>
  <properties>
    <hudson.model.ParametersDefinitionProperty>
```

```
      <parameterDefinitions>
        <hudson.model.StringParameterDefinition>
          <name>BRANCH</name>
          <description>Git branch to build</description>
          <defaultValue>main</defaultValue>
        </hudson.model.StringParameterDefinition>
      </parameterDefinitions>
    </hudson.model.ParametersDefinitionProperty>
  </properties>
  <scm class="hudson.plugins.git.GitSCM">
    <configVersion>2</configVersion>
    <userRemoteConfigs>
      <hudson.plugins.git.UserRemoteConfig>
        <url>https://gitlab.com/nasa/project.git</url>
      </hudson.plugins.git.UserRemoteConfig>
    </userRemoteConfigs>
    <branches>
      <hudson.plugins.git.BranchSpec>
        <name>*/${BRANCH}</name>
      </hudson.plugins.git.BranchSpec>
    </branches>
  </scm>
  <triggers>
    <hudson.triggers.SCMTrigger>
      <spec>H/5 * * * *</spec>
    </hudson.triggers.SCMTrigger>
  </triggers>
  <builders>
    <hudson.tasks.Maven>
      <targets>clean package</targets>
      <POM>pom.xml</POM>
    </hudson.tasks.Maven>
  </builders>
  <publishers>
    <hudson.plugins.git.GitPublisher>
      <configVersion>1</configVersion>
      <pushMerge>false</pushMerge>
      <pushTags>false</pushTags>
      <forcePush>false</forcePush>
      <gitUrl>https://gitlab.com/nasa/project.git</gitUrl>
      <credentialsId>gitlab-credentials</credentialsId>
    </hudson.plugins.git.GitPublisher>
  </publishers>
```

```
</project>
```

```
# .gitlab-ci.yml (or similar CI/CD file)

stages:
  - build
  - test
  - deploy

variables:
  ISS_COMM_SYSTEM_IMAGE: "iss-comm-system:latest"
  ISS_ENDPOINT: "https://iss.example.com/api/v1"

build:
  stage: build
  script:
    - docker build -t $ISS_COMM_SYSTEM_IMAGE .
  artifacts:
    paths:
      - docker-image-$ISS_COMM_SYSTEM_IMAGE.tar

test:
  stage: test
  script:
    - docker run -t $ISS_COMM_SYSTEM_IMAGE pytest
  artifacts:
    paths:
      - test-results.xml

deploy:
  stage: deploy
  script:
    - docker tag $ISS_COMM_SYSTEM_IMAGE
$ISS_ENDPOINT/$ISS_COMM_SYSTEM_IMAGE
    - docker push $ISS_ENDPOINT/$ISS_COMM_SYSTEM_IMAGE
    - curl -X POST -H "Content-Type: application/json" -d '{"image":
"'$ISS_COMM_SYSTEM_IMAGE'"}' $ISS_ENDPOINT/deploy
```

```
  only:
    - main
```

This code sends a **POST request** to the /deploy API endpoint with a JSON payload containing the image name and version. The Authorization header is set with a Bearer token, which should be replaced with a valid API token. The response status code is checked to ensure the deployment was successful.

```
import requests

iss_endpoint = "https://iss.example.com/api/v1"
api_token = "your_api_token_here"

headers = {
    "Authorization": f"Bearer {api_token}",
    "Content-Type": "application/json"
}

data = {"image": "iss-comm-system:latest"}

response = requests.post(f"{iss_endpoint}/deploy", headers=headers,
json=data)

if response.status_code == 201:
    print("Software deployed successfully!")
else:
    print("Error deploying software:", response.text)
```

CONFIG BASH FILES

Shell Script: to execute shell commands, such as deploying code to a staging environment

```
#!/bin/bash
ssh staging-server "cd /opt/nasa/project && git pull origin
${BRANCH}"
```

Email notifications

```
import javax.mail.*
import javax.mail.internet.*

def sendEmail(subject, body) {
  def mailServer = 'smtp.example.com'
  def fromAddress = 'nasa@example.com'
  def toAddress = 'qa@example.com'

  def message = new MimeMessage(session)
  message.setFrom(new InternetAddress(fromAddress))
  message.setRecipient(RecipientType.TO, new
InternetAddress(toAddress))
  message.setSubject(subject)
  message.setText(body)

  Transport.send(message)
}

sendEmail('Deployment Successful', 'The code has been deployed to
staging successfully.')
```

CONFIG ENV

```
# Set the default Java version
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64

# Set the default Maven version
export M2_HOME=/opt/maven
export PATH=$M2_HOME/bin:$PATH
```

```
# Set the default Git version
export GIT_HOME=/opt/git
export PATH=$GIT_HOME/bin:$PATH

# Set the default Jenkins version
export JENKINS_HOME=/var/lib/jenkins

# Set the default GitLab API URL
export GITLAB_API_URL=https://gitlab.com/api/v4

# Set the default GitLab username and token
export GITLAB_USERNAME=your-username
export GITLAB_TOKEN=your-token

# Set the default email configuration
export EMAIL_HOST=smtp.example.com
export EMAIL_PORT=587
export EMAIL_USERNAME=your-email-username
export EMAIL_PASSWORD=your-email-password
```

```
# Load software modules
source /usr/local/lib/global.profile

# Augment $PATH
PATH=$PATH:$HOME/bin:/u/scicon/tools/bin

# Set aliases
alias ll='ls -lrt'

# Set environment variables
export CFDROOT=/u/your_nas_username/bin
```

```
#!/bin/bash

# Run the Jenkins build
curl -X POST -u "$JENKINS_USERNAME:$JENKINS_PASSWORD"
"$JENKINS_URL/job/your-job-name/build"

# Check the build status
```

```
build_status=$(curl -s -u "$JENKINS_USERNAME:$JENKINS_PASSWORD"
"$JENKINS_URL/job/your-job-name/lastBuild/api/json" | jq -r '.result')

# Send an email notification based on the build status
if [ "$build_status" = "SUCCESS" ]; then
  echo "Build succeeded" | mail -s "ISS Communication System Build Result"
-r "nasa-dev-team@example.com" "nasa-dev-team@example.com"
elif [ "$build_status" = "FAILURE" ]; then
  echo "Build failed" | mail -s "ISS Communication System Build Result" -r
"nasa-dev-team@example.com" "nasa-dev-team@example.com"
fi
```

BB

- NASA's GitHub repository: https://github.com/nasa
- NASA's Docker Hub repository: https://hub.docker.com/u/nasa
- NASA's Kubernetes repository: https://github.com/nasa/kubernetes

Conclusion

This CI/CD plan outlines a comprehensive approach to implementing Continuous Integration and Continuous Deployment for the ISS Communication System - NASA using Jenkins and Docker. By following this plan, you can ensure the reliability, efficiency, and quality of the system, while also improving collaboration and reducing manual effort.