Project Report On

Deploying Application on HPC cluster by using internal CI-CD Pipeline

Submitted in partial fulfillment for the award of

Post Graduate Diploma in High Performance Computing System Administration from **C-DAC ACTS (Pune)**

Guided by

Mr.Pratik Maheshwari

Presented By

Mr. Sandeep Kamble	**PRN : 230940127048**
Mr. Pranav Ambhore	**PRN : 230940127044**
Mr. Mithlesh Murmu	**PRN : 230940127041**
Mr. Siddharth Birajdar	**PRN : 230940127012**

Centre of Development of Advanced Computing (C-DAC), Pune

सी-डैक CDAC ॲक्टस acts

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Mr. Sandeep Kamble	**PRN : 230940127048**
Mr. Pranav Ambhore	**PRN : 230940127044**
Mr. Mithlesh Murmu	**PRN : 230940127041**
Mr. Siddharth Birajdar	**PRN : 230940127012**

have successfully completed their project on

Deploying Application on HPC cluster by using internal CI-CD Pipeline

Under the Guidance of Mr. Pratik Maheshwari

Mr. Pratik Maheshwari

Project Guide **Project Supervisor**

HOD ACTS

Mr. Aditya Sinha

ACKNOWLEDGEMENT

This project **“Deploying Application on HPC cluster”** was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of **Mr. Pratik Maheshwari** for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to **Mr. Kaushal Sharma** (Manager (ACTS training Centre), C- DAC), for his guidance and support whenever necessary while doing this course **Post Graduate Diploma in High Performance Computing System Administration (PG- DHPCSA)** through C-DAC ACTS, Pune.

Our most heartfelt thank goes to **Ms. Swati Salunkhe** (Course Coordinator, PG-DHPCSA) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

Mr. Sandeep Kamble	PRN : 230940127048
Mr. Pranav Ambhore	PRN : 230940127044
Mr. Mithlesh Murmu	PRN : 230940127041
Mr. Siddharth Birajdar	PRN : 230940127012

Table of Contents

1. Abstract

Deploying applications on High-Performance Computing (HPC) clusters using Continuous Integration/Continuous Deployment (CI/CD) pipelines, such as Jenkins, involves a series of steps and practices designed to streamline the process of integrating new or updated code into a project, and ensuring that this code is deployed efficiently onto HPC systems. Here's a typical abstract for such a process:

"In the evolving landscape of software development, High-Performance Computing (HPC) applications demand robust, efficient, and agile deployment methodologies. Continuous Integration/Continuous Deployment (CI/CD) pipelines have emerged as pivotal in automating the software development lifecycle, enabling rapid integration, testing, and deployment of code. This study explores the implementation of CI/CD pipelines, with a focus on Jenkins, for deploying applications on HPC clusters. We outline a framework that leverages Jenkins to automate the build, test, and deployment phases, ensuring that code changes are seamlessly integrated and deployed to HPC environments. Our methodology addresses the unique challenges posed by HPC systems, including environment configuration, dependency management, and scalability. Through a case study, we demonstrate the effectiveness of our approach in improving deployment frequency, reducing integration errors, and enhancing overall software quality. Our findings suggest that integrating CI/CD pipelines with HPC deployments accelerates development cycles, optimizes resource utilization, and fosters a culture of continuous improvement among development teams. This research contributes to the field by providing insights and practical guidelines for organizations seeking to enhance their HPC application deployment processes through the adoption of CI/CD practices."

This abstract synthesizes the objectives, methodology, findings, and implications of deploying applications on HPC clusters using a CI/CD pipeline, specifically through Jenkins. It highlights the benefits of such an approach, including improved deployment frequency, reduced errors, and better resource utilization.

2. Introduction

This Project aims to implement Continuous Integration and Continuous Deployment using Jenkins to Deploy Applications on High Performance Computing Cluster.
This Project makes use of jenkins for Continuous Integration and Continuous Deployment , xcat for Node Provisioning ,slurm for Resource Management, iptables for routing rules ,python program to run on HPC cluster using Jenkins.

In the realm of software development, system administration, and scientific computing, tools such as Jenkins, xCAT, Slurm, iptables, and the Python programming language play pivotal roles in automating, managing, and securing processes and computations. Each of these tools offers a specialized set of functionalities designed to streamline operations, enhance productivity, and bolster security. This introduction provides a brief overview of each component and its significance in modern computing environments.

Jenkins

Jenkins is an open-source automation server that facilitates Continuous Integration and Continuous Deployment (CI/CD) practices. It enables developers to build, test, and deploy their software automatically, thereby increasing efficiency and reducing the likelihood of errors. Jenkins supports a wide array of plugins that extend its capabilities, making it highly customizable and adaptable to various development needs. It serves as a central platform for automating code integration and deployment tasks, making the software development process more agile and responsive to changes.

xCAT

Extreme Cloud Administration Toolkit (xCAT) is a scalable distributed computing management and provisioning tool. It is designed to deploy and manage large numbers of systems efficiently, including HPC clusters, cloud environments, and virtualized data centers. xCAT offers a comprehensive suite of features for hardware management, operating system deployment, network configuration, and security management. It is particularly favored in environments where managing a large scale of systems is required, offering automation and simplification of complex administrative tasks.

NFS

Network File System (NFS) is a distributed file system protocol that allows a user on a client computer to access files over a network in a manner similar to how local storage is accessed. NFS operates on a server-client architecture where the server hosts the files to be shared, and the client mounts the server's file system within its own directory structure. Once mounted, the client can interact with the remote file system as if it were a local file system, performing operations such as reading, writing, and executing files.NFS uses Remote Procedure Calls (RPC) to facilitate communication between the client and server.

Slurm

Slurm (Simple Linux Utility for Resource Management) is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for Linux clusters. It is widely used in HPC environments to allocate resources, manage job queues, and schedule job execution, optimizing the utilization of available computing resources. Slurm ensures that the computational workloads are efficiently distributed across the cluster's nodes, enhancing the overall performance and throughput of computational tasks.

iptables

iptables is a command-line firewall utility that allows system administrators to configure the IP packet filter rules implemented by the Linux kernel's netfilter framework. It provides powerful capabilities for controlling network traffic to and from a system, including packet filtering, network address translation (NAT), and port redirection. iptables is essential for securing network interfaces on Linux systems, allowing administrators to define rules that permit or block traffic based on IP addresses, port numbers, and protocols.

Python Programming

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming, making it suitable for a wide range of applications. Python's extensive standard library and the vast ecosystem of third-party packages enable developers to tackle tasks in web development, data analysis, artificial intelligence, scientific computing, and more. Its ease of learning and broad applicability have made Python one of the most popular programming languages in the world.

The integration and utilization of Jenkins, xCAT, Slurm, iptables, and Python in computing environments exemplify the advancement and specialization in software development, system administration, and scientific computing. Each tool offers unique capabilities that, when leveraged effectively, can significantly enhance the efficiency, security, and performance of computing operations. Whether it's automating software deployment, managing large-scale clusters, scheduling computational jobs, securing network traffic, or developing versatile applications, these tools collectively contribute to the sophistication and dynamism of modern computing practices.

3. Implementing CI-CD Pipeline to deploy app on HPC cluster

For implementing CI-CD Pipeline to deploy app on HPC cluster we needed Jenkins to work on one machine which will deploy application on HPC cluster.

While HPC cluster had to be made for that. To make HPC cluster we needed xcat to provision node OS and then perform routing for the nodes via xcat machine (here we call it HPC Master) while its nodes are compute nodes.

Here are the steps to implement SSDLC in DevSecOps:

1. Plan:
   - In this stage, we identified that we needed Jenkins Machine (Production Machine) which will deploy applications on Hpc cluster.
   - Hpc cluster which would be made by xcat node provisioning and its resources had to be managed for that we decide to use Slurm. The routing between these components was necessary for that we decided to use Iptabes.Finally one application to deploy had to be found out.

2. Design:
   - In this stage, we designed the architecture of our working cluster and number of machines needed.
   - In our case we needed one Production Machine ,one HPC Master which would be having xcat to boot compute nodes . Number of compute nodes were decided to be only two.
   - Hpc master would also be performing resource management via Slurm.

3. Develop:
   - In this stage, we chose to go with CentOS 7 for Hpc Master ,Jenkins machine and compute nodes.
   - We first Provisioned two nodes OS via xCat ,then configured routing using Iptables between Hpc Master and Compute nodes to outside world.
   - Here Hpc master acted as a gateway to compute nodes. Also configured ssh between from Hpc master to compute nodes.
   - Then we configured Jenkins Machine which will deploy applications on Hpc cluster. This machine will do so by pipeline by ssh for this ssh had to enabled between Jenkins machine and Hpc Master so we did that.

- Configured common storage for Hpc Cluster so that files remain same everywhere via NFS
- Finally we configured Slurm on Hpc Master and compute nodes by its respective daemons and configureations for Hpc master as controller and Compute nodes and worker nodes.

4. Test:
   - In this stage, we first tested if routing is done properly between hpc master and compute nodes and if compute ndoes are able to access outside internet via Hpc master as gateway by pinging between Hpc master and compute nodes and pinging from compute nodes to outside internet example- Ping google.
   - Tested ssh between Jenkins to Hpc Master, ssh between Hpc Master and compute nodes .
   - Then tested if some simple job via Jenkins works on Hpc Master
   - Tested if slurm is able to run jobs on compute nodes .

5. Deploy:
   - In this stage, we finally deployed python application via Jenkins machine which first pulled the app from github repository to hpc machine nfs storage and then in next stage Jenkins machine submitted batch job to run the python program on specified nodes.

6. Operate:
   - In this stage, we need monitored the CI-CD and Hpc Environment if its working fine.

By following these steps, we Implemented CI-CD infrastructure to facilitate deploying of applications on HPC cluster.

4. System Requirement

User Interface

1. Jenkins Machine on Cent OS 7

Software Requirement

1. GIT
2. Jenkins
3. xCat
4. Iptables
5. Slurm
6. Nfs

Hardware Requirement

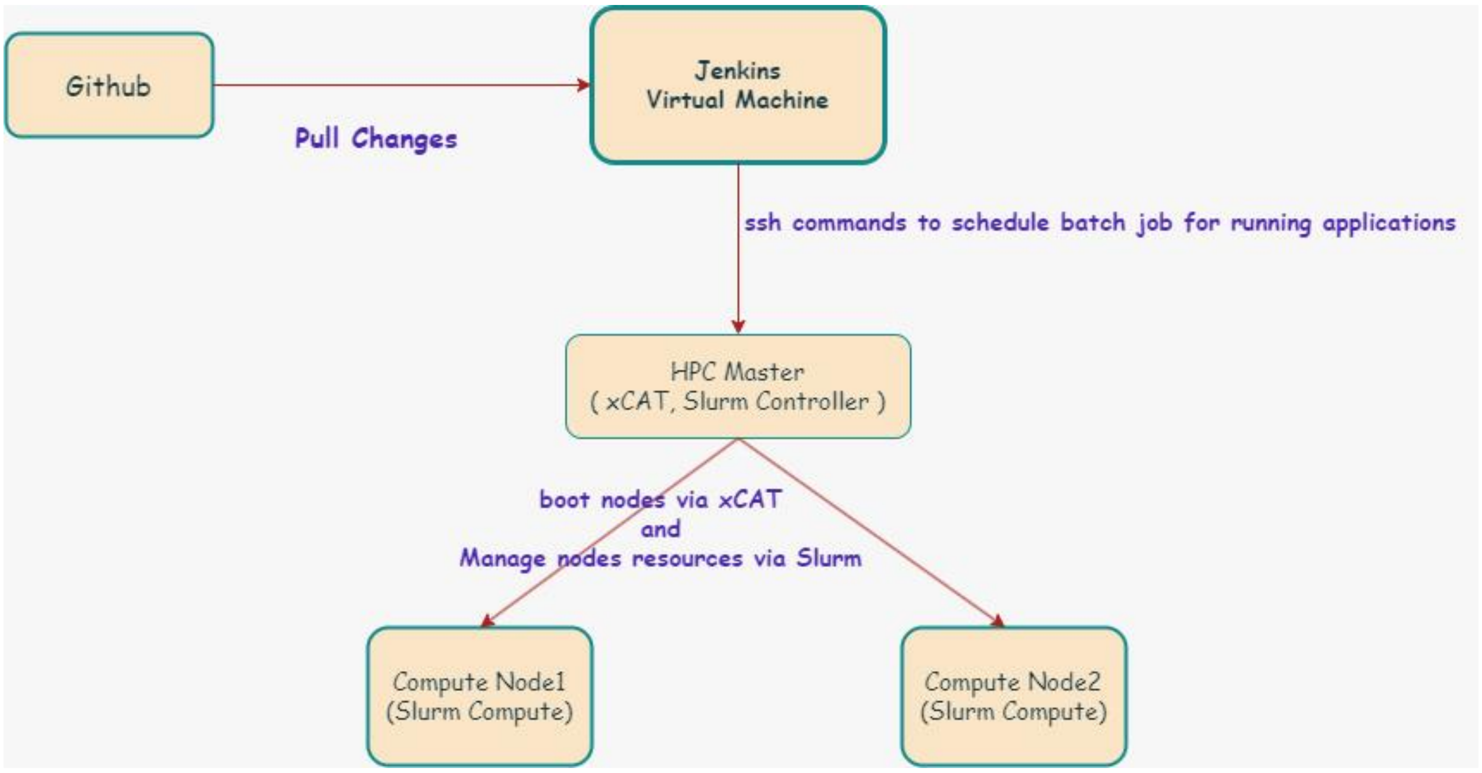
1. Cent OS 7 30GB HD, 4 GB RAM

Used Languages

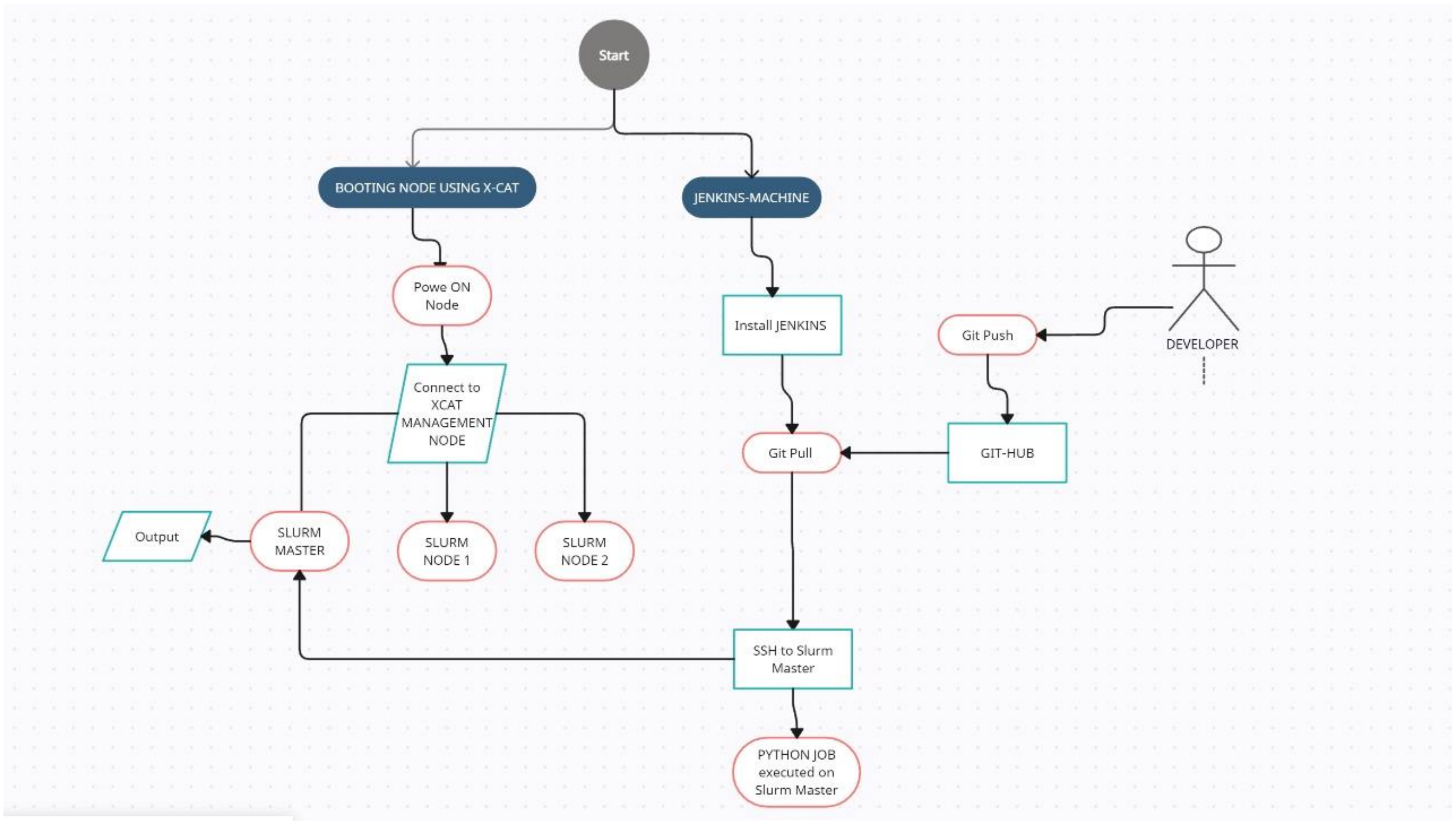
1. Python
2. Shell Scripting

5. System Architecture

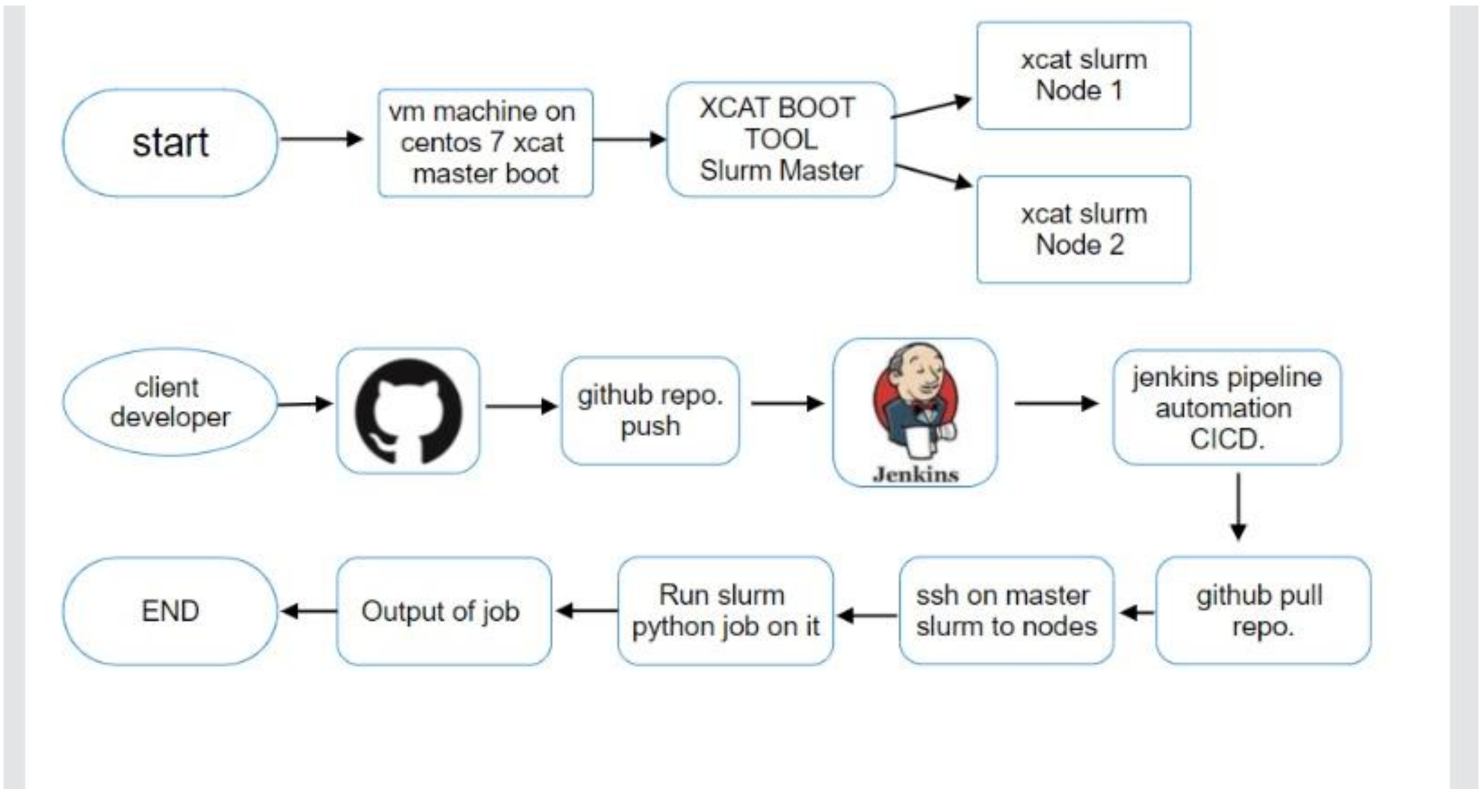
Data Flow Diagram


Github
Pull Changes
Jenkins Virtual Machine
ssh commands to schedule batch job for running applications
HPC Master (xCAT, Slurm Controller)
boot nodes via xCAT and Manage nodes resources via Slurm
Compute Node1 (Slurm Compute)
Compute Node2 (Slurm Compute)

6. Activity Diagram



7. Process Flow Diagram


start
vm machine on centos 7 xcat master boot
XCAT BOOT TOOL Slurm Master
xcat slurm Node 1
xcat slurm Node 2
client developer
github repo. push
Jenkins
jenkins pipeline automation CICD.
github pull repo.
ssh on master slurm to nodes
Run slurm python job on it
Output of job
END

8. xCat Node Provisioning

xCAT (**Extreme Cloud Administration Toolkit**) is open-source distributed computing management software developed by IBM, used for the deployment and administration of Linux or AIX based clusters.

xCAT can:

- Create and manage diskless clusters
- Install and manage many Linux cluster machines (physical or virtual) in parallel
- Set up a high-performance computing software stack, including software for batch job submission, parallel libraries, and other software that is useful on a cluster
- Cloning and imaging Linux and Windows machines

xCAT has specific features designed to take advantage of IBM hardware including:

- Remote Power Control
- Remote POST/BIOS console
- Serial over LAN functions
- Hardware alerts and vitals provided via SNMP and email
- Inventory and hardware management

xCAT achieved recognition in June 2008 for having been used with the IBM Roadrunner, which set a computing speed record at that time.[3][4]

xCAT is the default systems management tool of the IBM Intelligent Cluster solution.

xCAT enables the administrator to:

1. Discover the hardware servers
2. Execute remote system management
3. Provision operating systems on physical or virtual machines
4. Provision machines in Diskful (stateful) and Diskless (stateless)
5. Install and configure user applications
6. Parallel system management
7. Integrate xCAT in Cloud

You’ve reached xCAT documentation site, The main page product page is http://xcat.org

xCAT is an open source project hosted on GitHub. Go to GitHub to view the source, open issues, ask questions, and participate in the project

Extreme Cloud/Cluster Administration Toolkit (xCAT) is a comprehensive and powerful tool for managing and provisioning both small and large-scale High-Performance Computing (HPC) clusters, cloud environments, and virtualized data centers. xCAT is known for its scalability, flexibility, and extensive support for different hardware, operating systems, and virtualization platforms. It offers a wide range of features for hardware management, operating system deployment, network configuration, and security management.

Key Features of xCAT

- Scalability: xCAT can manage from a handful to thousands of nodes.
- Heterogeneity: Supports various hardware, operating systems (Linux, Windows), and virtualization platforms (VMware, KVM, PowerVM).
- Automation: Automates the process of operating system installation, configuration, and firmware updates.
- Customization: Allows for detailed customization of the provisioning process through templates and scripts.
- Network Configuration: Manages complex network configurations, including DHCP, DNS, and PXE boot environments.
- Monitoring and Management: Provides tools for monitoring system status and performing remote management operations.

xCAT Architecture

- Management Node (MN): The central node that controls the xCAT environment, managing the configuration and deployment of service and compute nodes.
- Service Nodes (SN): Optional nodes that help scale the management capabilities of xCAT, especially in large environments, by providing local management services to compute nodes.
- Compute Nodes: The target nodes being managed and provisioned by xCAT.

Key xCAT Commands

xCAT commands are executed from the Management Node and can be categorized into various functionalities:

- **Node Deployment and Management**
- `nodeset`: Sets the boot options for nodes, such as installing an OS or booting into a diskless environment.
- `nodestat`: Displays the status of nodes, showing if they are up, down, or unreachable.
- `rinstall`/`rsetboot`: Initiates the OS installation process or sets the boot device for the nodes.

- **Network Management**
- `makedhcp`: Configures DHCP settings for nodes to enable network boot.
- `makehosts`: Updates the `/etc/hosts` file to ensure hostname resolution for xCAT nodes.
- `makenetworks`: Manages network definitions within xCAT.

- **Configuration Management**
- `chdef`: Changes or sets attributes for xCAT objects like nodes, groups, networks, etc.
- `lsdef`: Lists definitions of xCAT objects to view their current configurations.
- `mkdef`: Creates new definitions for xCAT objects.

- **Software and Firmware Management**
- `updatenode`: Updates or installs software on nodes.
- `rflash`: Used for updating firmware on nodes.

- **System Monitoring and Remote Execution**
- `xdsh`: Executes commands on one or more nodes remotely.
- `xcatprobe`: A diagnostic tool to check the health and configuration of the xCAT cluster.

Configuration and Setup

The setup and configuration of xCAT involve installing the xCAT package on the Management Node, configuring network services like DHCP, DNS, and TFTP (for network boot environments), and defining the nodes and their attributes within xCAT. The process typically follows these steps:

1. Install xCAT: Download and install the xCAT package on the Management Node.
2. Define Networks: Use `makenetworks` to define network settings.
3. Define Nodes: Use `mkdef` to create node objects and specify their attributes, including hardware specifications, network settings, and OS image to be deployed.
4. Set Up DHCP, DNS, and TFTP: Configure these services using xCAT commands like `makedhcp` and `makehosts` to support network boot and provisioning.
5. Deploy Nodes: Use commands like `nodeset` and `rinstall` to initiate the OS deployment process.

xCAT is a powerful tool for managing complex and scalable computing environments. Its command-line interface provides administrators with detailed control over the provisioning process, network configuration, and system management tasks. Through its extensive support for various hardware and software platforms, xCAT enables efficient and flexible management of HPC clusters and cloud environments.

9. Jenkins server configuration

Jenkins is used to continually create and test software projects, making it easier for developers and DevOps engineers to integrate changes to the project and for consumers to get a new build.
It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration, and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase, and RTC

Project setup with CI CD Pipeline:

Pre-requisites:

1. **GitHub**
   `( Repo url: https://github.com/Sandip1717/HPC-SLURM.git/)`
2. **Slurm**
3. **HPC Cluster**

Installing Jenkins as a service on CentOS 7 involves several steps, from adding the Jenkins repository and installing Jenkins to starting the Jenkins service and ensuring it runs at boot. Jenkins is a popular open-source automation server that facilitates continuous integration and continuous deployment (CI/CD) practices in software development. Here's a detailed guide on how to install Jenkins on CentOS 7:

Step 1: Install Java

Jenkins is a Java-based application, so Java needs to be installed on your system. Jenkins requires Java 8 or Java 11. Here's how to install Java 8 using the `yum` package manager:

```
sudo yum install java-1.8.0-openjdk-devel
```

After installing, verify the Java installation by checking the version:

```
java -version
```

Step 2: Add the Jenkins Repository

Before installing Jenkins, you need to add the Jenkins repository to your system. This ensures you get the latest version of Jenkins directly from the official source.

1. Import the Jenkins repository GPG keys:

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
```

2. Add the Jenkins repository to your system:

```
sudo curl --create-dirs -o /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

Step 3: Install Jenkins

Now that the Jenkins repository is added to your system, you can install Jenkins using `yum`:

```
sudo yum install jenkins
```

Step 4: Start and Enable Jenkins Service

After installation, you need to start the Jenkins service and enable it to launch at system boot:

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Step 5: Open Firewall Ports for Jenkins

Jenkins runs on port 8080 by default. If you have a firewall running (like firewalld), you need to open this port to allow traffic to Jenkins:

```
sudo firewall-cmd --permanent --zone=public --add-port=8080/tcp
sudo firewall-cmd --reload
```

Step 6: Accessing Jenkins

Once Jenkins is installed and the firewall is configured, you can access the Jenkins web interface by navigating to `http://your_server_ip:8080` in your web browser. The first time you access Jenkins, you will be asked to unlock it using an initial admin password.

1. Retrieve the initial admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

2. Copy the password and paste it into the Jenkins setup wizard in your browser to unlock Jenkins.

3. Follow the on-screen instructions to complete the installation, which includes installing suggested plugins and creating an admin user.

Step 7: Configure Jenkins (Optional)

After installation, you might want to configure Jenkins according to your project needs. This can include configuring system settings, managing plugins, creating jobs/projects, and setting up build triggers.

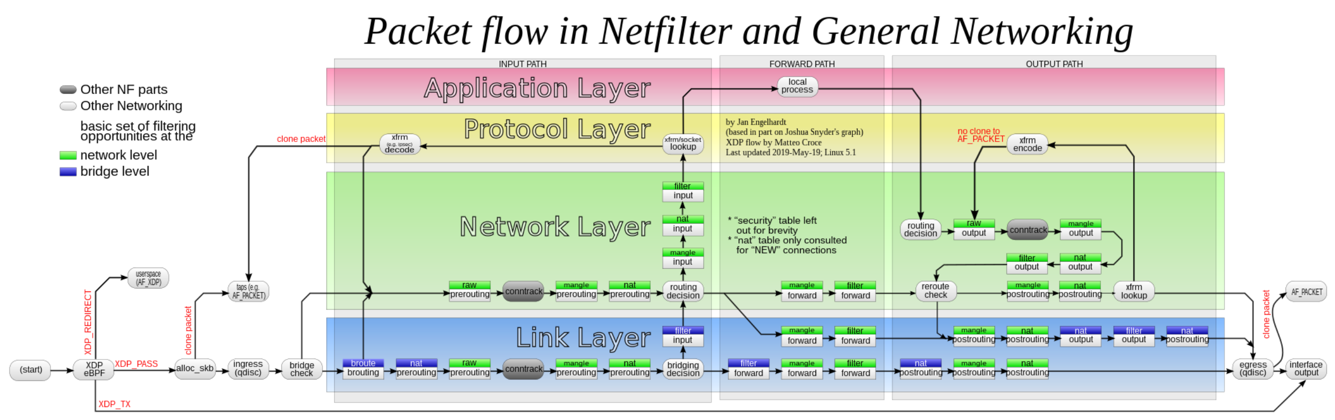
10. Routing Configuration

iptables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules. The filters are organized in different tables, which contain chains of rules for how to treat network traffic packets. Different kernel modules and programs are currently used for different protocols; *iptables* applies to IPv4, *ip6tables* to IPv6, *arptables* to ARP, and *ebtables* to Ethernet frames.

iptables requires elevated privileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages, which can be opened using `man iptables` when installed. It may also be found in `/sbin/iptables`, but since iptables is more like a service rather than an "essential binary", the preferred location remains /usr/sbin.

The term *iptables* is also commonly used to inclusively refer to the kernel-level components. *x_tables* is the name of the kernel module carrying the shared code portion used by all four modules that also provides the API used for extensions; subsequently, *Xtables* is more or less used to refer to the entire firewall (v4, v6, arp, and eb) architecture.

iptables allows the system administrator to define *tables* containing *chains* of *rules* for the treatment of packets. Each table is associated with a different kind of packet processing. Packets are processed by sequentially traversing the rules in chains. A rule in a chain can cause a goto or jump to another chain, and this can be repeated to whatever level of nesting is desired. (A jump is like a “call”, i.e. the point that was jumped from is remembered.) Every network packet arriving at or leaving from the computer traverses at least one chain.



Packet flow paths. Packets start at a given box and will flow along a certain path, depending on the circumstances.

The origin of the packet determines which chain it traverses initially. There are five *predefined chains* (mapping to the five available Netfilter hooks), though a table may not have all chains. Predefined chains have a *policy*, for example DROP, which is applied to the packet if it reaches the end of the chain. The system administrator can create as many other chains as desired. These chains have no policy; if a packet reaches the end of the chain it is returned to the chain which called it. A chain may be empty.

- `PREROUTING`: Packets will enter this chain before a routing decision is made.
- `INPUT`: Packet is going to be locally delivered. It does not have anything to do with processes having an opened socket; local delivery is controlled by the "local-delivery" routing table: `ip route show table local`.
- `FORWARD`: All packets that have been routed and were not for local delivery will traverse this chain.
- `OUTPUT`: Packets sent from the machine itself will be visiting this chain.
- `POSTROUTING`: Routing decision has been made. Packets enter this chain just before handing them off to the hardware.

A chain does not exist by itself; it belongs to a *table*. There are three tables: *nat*, *filter*, and *mangle*. Unless preceded by the option *-t*, an `iptables` command concerns the *filter* table by default. For example, the command `iptables -L -v -n`, which shows some chains and their rules, is equivalent to `iptables -t filter -L -v -n`. To show chains of table *nat*, use the command `iptables -t nat -L -v -n`

Each rule in a chain contains the specification of which packets it matches. It may also contain a *target* (used for extensions) or *verdict* (one of the built-in decisions). As a packet traverses a chain, each rule in turn is examined. If a rule does not match the packet, the packet is passed to the next rule. If a rule does match the packet, the rule takes the action indicated by the target/verdict, which may result in the packet being allowed to continue along the chain or may not. Matches make up the large part of rulesets, as they contain the conditions packets are tested for. These can happen for about any layer in the OSI model, as with e.g. the `--mac-source` and `-p tcp --dport` parameters, and there are also protocol-independent matches, such as `-m time`.

The packet continues to traverse the chain until either

1. a rule matches the packet and decides the ultimate fate of the packet, for example by calling one of the `ACCEPT` or `DROP`, or a module returning such an ultimate fate; or
2. a rule calls the `RETURN` verdict, in which case processing returns to the calling chain; or
3. the end of the chain is reached; traversal either continues in the parent chain (as if `RETURN` was used), or the base chain policy, which is an ultimate fate, is used.

Targets also return a verdict like `ACCEPT` (`NAT` modules will do this) or `DROP` (e.g. the `REJECT` module), but may also imply `CONTINUE` (e.g. the `LOG` module; `CONTINUE` is an internal name) to continue with the next rule as if no target/verdict was specified at all.

`iptables` is a versatile and powerful tool that acts as a firewall for managing network traffic in Linux-based systems. It uses tables to organize sets of rules, called chains, to determine how to treat network packets. The three primary tables are `filter`, `nat`, and `mangle`, but we'll focus mainly on `filter` for basic firewall functionalities and `nat` for network address translation, which are pertinent to your query about enabling forwarding, NAT rules, and allowing every input in the input chain.

- **Starting with iptables**

  Before diving into configurations, ensure `iptables` is installed on your system. It typically comes pre-installed on many Linux distributions. You can verify its presence and version with:

  ```
  iptables --version
  ```

- **Basic Concepts**

  - **Tables**: Containers for several chains.
  - **Chains**: Sets of rules that define what action to take on packets.
  - **Rules**: Conditions that if met, trigger an action (e.g., accept, drop, forward).

- **Key Tables and Chains**

  - **Filter Table** The default table, which contains the `INPUT`, `OUTPUT`, and `FORWARD` chains for packet filtering.
  - **NAT Table**: Used for network address translation, containing `PREROUTING`, `POSTROUTING`, and `OUTPUT` chains.

- **Enabling IP Forwarding**

  To route packets between interfaces on a Linux system (a requirement for functioning as a router or gateway), you must enable IP forwarding:

  ```
  echo 1 > /proc/sys/net/ipv4/ip_forward
  ```

  Or make it permanent by editing `/etc/sysctl.conf`:

  ```
  net.ipv4.ip_forward = 1
  ```

  And apply the changes:

  ```
  sysctl -p
  ```

Configuring NAT and Forwarding Rules

- **Setting Up NAT**

  To set up NAT (e.g., masquerading), which allows devices on a private network to connect to the internet through a single public IP address, use the `POSTROUTING` chain in the `nat` table:

  ```
  iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
  ```

  This command appends a rule to masquerade outgoing packets from the private network interface (assuming `eth0` is your public-facing interface).

- **Enabling Packet Forwarding**

  To allow forwarding from one network interface to another (e.g., from `eth1` (internal) to `eth0` (external)), you might need to add forwarding rules:

  ```
  iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
  iptables -A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
  ```

  These rules enable forwarding from the internal to the external network and vice versa for established connections.

- **Allowing Every Input in the Input Chain**

  To accept all incoming traffic (be cautious with this setting as it can expose your system to vulnerabilities), you can set the default policy for the `INPUT` chain to `ACCEPT`:

  ```
  iptables -P INPUT ACCEPT
  ```

  Or, to explicitly add a rule to accept all incoming traffic:

  ```
  iptables -A INPUT -j ACCEPT
  ```

- **Saving and Persisting iptables Rules**

  Rules created with `iptables` are not saved automatically through reboots. To save and restore them, use `iptables-save` and `iptables-restore` respectively, or utilize a package like `iptables-persistent` on Debian-based systems.

`iptables` provides comprehensive capabilities for managing network traffic and implementing security policies on Linux systems. When configuring `iptables`, it's crucial to proceed with caution, as incorrect settings can lead to security vulnerabilities or disrupt network connectivity. Always test configurations in a safe environment before applying them to production systems.

11. NFS Configuration

Network File System (**NFS**) is a distributed file system protocol originally developed by Sun Microsystems (Sun) in 1984, allowing a user on a client computer to access files over a computer network much like local storage is accessed. NFS, like many other protocols, builds on the Open Network Computing Remote Procedure Call (ONC RPC) system. NFS is an open IETF standard defined in a Request for Comments (RFC), allowing anyone to implement the protocol.

Version 2 of the protocol (defined in RFC 1094, March 1989) originally operated only over User Datagram Protocol (UDP). Its designers meant to keep the server side stateless, with locking (for example) implemented outside of the core protocol. People involved in the creation of NFS version 2 include Russel Sandberg, Bob Lyon, Bill Joy, Steve Kleiman, and others.

The Virtual File System interface allows a modular implementation, reflected in a simple protocol. By February 1986, implementations were demonstrated for operating systems such as System V release 2, DOS, and VAX/VMS using Eunice. NFSv2 only allows the first 2 GB of a file to be read due to 32-bit limitations.

Starting with NFS

To use NFS, you need to set up an NFS server that will share the files, and NFS clients that will access those files.

On the Server Side

1. **Install NFS Server**: On a Linux system, you typically install the NFS server package. For example, on Ubuntu or Debian-based systems, you can use:

```
sudo apt update
sudo apt install nfs-kernel-server
```

2. **Configure Exports**: Decide which directories you want to share over the network and add them to the `/etc/exports` file. This file dictates which directories are shared, and with whom they can be shared. For example, to share `/var/nfs` directory with a client (`192.168.1.5`) with read-write access, you'd add the following line:

```
   /var/nfs 192.168.1.5(rw,sync,no_root_squash,no_subtree_check)
```

   - `rw`: Read and write permissions.
   - `sync`: Reply to requests only after the changes have been committed to stable storage.
   - `no_root_squash`: Prevents mapping root user to the `nobody` user.
   - `no_subtree_check`: Prevents subtree checking.

3. **Export the Shared Directories**: After configuring the directories you want to share, export them:

```
   sudo exportfs -a
```

4. **Start the NFS Server**: Ensure the NFS server is running. On systems using `systemd`, you can use:

```
   sudo systemctl start nfs-kernel-server
```

   And enable it to start at boot:

```
   sudo systemctl enable nfs-kernel-server
```

On the Client Side

1. **Install NFS Client**: On the client machine, you need to install NFS client utilities. For Ubuntu or Debian:

```
sudo apt update
sudo apt install nfs-common
```

2. **Mount the NFS Share**: Create a mount point and mount the shared directory from the server. For example, to mount the `/var/nfs` directory shared by the server (`192.168.1.10`) to a local directory (`/mnt/nfs`):

```
sudo mkdir -p /mnt/nfs
sudo mount 192.168.1.10:/var/nfs /mnt/nfs
```

Using NFS

Once mounted, the client can access the NFS shared directory as if it were a local filesystem, enabling file operations like reading, writing, and executing based on the permissions set by the NFS server.

Configuration and Commands

- **/etc/exports**: Main configuration file on the server for defining shared directories.
- **exportfs**: Utility for managing exported directories. Useful commands include:
- `**sudo exportfs -a**`: Export all shared directories.
- `**sudo exportfs** -r`: Re-export all directories, applying any changes.
- `**sudo exportfs -v**`: Display currently exported directories and their parameters.
- **showmount**: Utility for displaying mounted directories. Common usage:
- `**showmount -e** [server]`: Show the list of exports available on the server.
- **mount and umount**: Standard Unix commands for mounting and unmounting filesystems, including NFS shares.

Best Practices and Tips

- **Security**: Consider using NFSv4 for better security features like Kerberos authentication.
- **Firewall Configuration**: Ensure that the necessary ports are open on both the server and client firewalls to allow NFS traffic.
- **Network Stability**: NFS relies on a stable network connection. Network disruptions can lead to stalled mounts or data inconsistency.
- **Performance Tuning**: NFS performance can be tuned by adjusting parameters like `rsize` and `wsize` for read and write block sizes.

NFS provides a simple and effective way to share files across a network, but proper setup and configuration are crucial for ensuring secure and efficient operation.

12. Slurm Configurations

The **Slurm Workload Manager**, formerly known as **Simple Linux Utility for Resource Management** (**SLURM**), or simply **Slurm**, is a free and open-source job scheduler for Linux and Unix-like kernels, used by many of the world's supercomputers and computer clusters.

It provides three key functions:

- allocating exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work,
- providing a framework for starting, executing, and monitoring work, typically a parallel job such as Message Passing Interface (MPI) on a set of allocated nodes, and
- arbitrating contention for resources by managing a queue of pending jobs.

Slurm is the workload manager on about 60% of the TOP500 supercomputers.[1]

Slurm uses a best fit algorithm based on Hilbert curve scheduling or fat tree network topology in order to optimize locality of task assignments on parallel computers.[2]

Slurm's design is very modular with about 100 optional plugins. In its simplest configuration, it can be installed and configured in a couple of minutes. More sophisticated configurations provide database integration for accounting, management of resource limits and workload prioritization.

Slurm features include

- No single point of failure, backup daemons, fault-tolerant job options

- Highly scalable (schedules up to 100,000 independent jobs on the 100,000 sockets of IBM Sequoia)
- High performance (up to 1000 job submissions per second and 600 job executions per second)
- Free and open-source software (GNU General Public License)
- Highly configurable with about 100 plugins
- Fair-share scheduling with hierarchical bank accounts
- Preemptive and gang scheduling (time-slicing of parallel jobs)
- Integrated with database for accounting and configuration
- Resource allocations optimized for network topology and on-node topology (sockets, cores and hyperthreads)
- Advanced reservation
- Idle nodes can be powered down
- Different operating systems can be booted for each job
- Scheduling for generic resources (e.g. Graphics processing unit)
- Real-time accounting down to the task level (identify specific tasks with high CPU or memory usage)
- Resource limits by user or bank account

Installing and configuring Slurm (Simple Linux Utility for Resource Management) involves several steps, from setting up the necessary prerequisites to adjusting configuration files for your specific cluster environment. This guide provides a comprehensive overview of the process, focusing on a basic setup that can be expanded or modified based on specific requirements.

Step 1: Prerequisites

Before installing Slurm, ensure your system meets the following prerequisites:

- A Linux distribution (e.g., CentOS, Ubuntu, Debian)
- Network connectivity between all nodes in the cluster
- Time synchronization across the cluster (e.g., using NTP)
- Unique hostnames for each node
- A designated control node (also can serve as a compute node)
- Optional: A MySQL or MariaDB database for Slurm accounting and job history (for larger or more complex setups)

Step 2: Installation

You can install Slurm from source or use pre-built packages available for many Linux distributions.

- Installing from Packages

For Debian-based distributions (like Ubuntu):

```
sudo apt update
sudo apt install slurm-wlm
```

For Red Hat-based distributions (like CentOS):

```
sudo yum install slurm slurm-example-configs
```

- Installing from Source

1. Download the latest Slurm source from the official [Slurm website](https://www.schedmd.com/downloads.php).
2. Extract the tarball and compile Slurm:

```
tar -xzf slurm-<version>.tar.bz2
cd slurm-<version>
./configure
make
sudo make install
```

3. Optionally, install the Slurm plugins:

```
cd contribs
sudo make install
```

Step 3: Configuration

The main configuration file for Slurm is `slurm.conf`. This file must be identical on all nodes (control, compute, and login nodes).

1. Create `slurm.conf`: You can start with an example configuration file typically found in `/usr/share/doc/slurm-wlm` or `/usr/share/doc/slurm-<version>` directory. Alternatively, use the online [Slurm Configuration Tool](https://slurm.schedmd.com/configurator.html) to generate a `slurm.conf` based on your cluster's specifications.

2. Edit `slurm.conf`: Open the `slurm.conf` file in a text editor and modify it according to your cluster's setup. Key parameters include:

- `ControlMachine`: The hostname of the control node.
- `NodeName`: Defines compute nodes, their resources (CPUs, memory, etc.), and properties.
- `PartitionName`: Defines partitions (queues) that organize sets of nodes for job allocation.
- `SlurmctldPort` and `SlurmdPort`: Ports for Slurm control and compute daemons.
- Configure additional settings as needed, such as `JobAcctGatherType` for job accounting, `SchedulerType` for the scheduling algorithm, etc.

3. Distribute `slurm.conf`: Ensure that `slurm.conf` is copied to the same location (e.g., `/etc/slurm/`) on all nodes in the cluster.

Step 4: Starting Slurm Daemons

1. Control Node: Start the `slurmctld` daemon:

```
sudo systemctl enable slurmctld
sudo systemctl start slurmctld
```

2. Compute Nodes: Start the `slurmd` daemon on each compute node:

```
sudo systemctl enable slurmd
sudo systemctl start slurmd
```

3. (Optional) SlurmDBD: If using SlurmDBD for accounting, configure `slurmdbd.conf`, start the `slurmdbd` daemon, and ensure `slurm.conf` includes the correct `AccountingStorage` settings.

Step 5: Verify Installation

After starting the daemons, verify that Slurm is running correctly:

- Check the status of the control and compute daemons:

```
sudo systemctl status slurmctld slurmd
```

- Use Slurm commands to check the cluster's status:

```
sinfo  # View status of partitions and nodes
squeue  # View job queue
```

- **Starting with Slurm**

Setting up Slurm involves configuring a Slurm controller (slurmctld), Slurm nodes (slurmd), and optionally, SlurmDBD for accounting purposes. Here's a simplified overview of how to start with Slurm:

- *Installation*

1. Install Slurm: Installation methods vary depending on the Linux distribution. Many distributions offer Slurm packages through their package managers. Alternatively, Slurm can be compiled from source.

- *Configuration*

2. Configure Slurm: The main configuration file for Slurm is `slurm.conf`, typically located in `/etc/slurm/`. This file contains the cluster configuration, including control and compute nodes definitions, scheduling options, and other settings.

- Control Machine: Define the control machine with `ControlMachine` directive.

- Node Definitions: Define compute nodes, their resources (CPUs, memory), and partitions (queues).
- Scheduling Options: Configure scheduling policies according to your needs.

3. Start Slurm Services:
- On the control machine, start `slurmctld` (the Slurm controller daemon).
- On each compute node, start `slurmd` (the Slurm daemon).

4. (Optional) SlurmDBD: If using SlurmDBD for accounting, configure and start the SlurmDBD daemon.

- **Using Slurm**

Once Slurm is configured and running, users can submit jobs, and administrators can manage the cluster resources.

1. Basic Commands for Users

- sinfo: Displays information about cluster nodes and partitions.
- squeue: Lists jobs queued and running in the system.
- sbatch: Submits a batch script to the scheduler. For example:

sbatch my_job_script.sh

- scancel: Cancels a queued or running job.
- srun: Executes a command directly via Slurm, often used for interactive jobs.

#### *Sample Batch Script*
```bash
#!/bin/bash
#SBATCH --job-name=test_job
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --partition=standard

srun hostname
srun sleep 60
```

This script submits a job named `test_job` to the `standard` partition, requesting one task, with a time limit of 10 minutes, and executes two commands: `hostname` and `sleep 60`.

2. Basic Commands for Administrators

- scontrol: Utility for administrative tasks such as updating configuration, managing jobs, and nodes status.
- sacct: Displays accounting data for jobs if SlurmDBD is configured.
- sacctmgr: Manages Slurm accounts, users, and associations.

- Configuration Details

- slurm.conf: The primary configuration file, which includes details about nodes, partitions, and scheduling policies. Each node and partition definition allows you to specify various parameters to control resource allocation and job scheduling.
- cgroup.conf: Optional configuration for controlling resource limits via Linux control groups.

13. Application Integration and Git

Git Overview

Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is used to track changes in source code during software development, enabling multiple developers to work together on non-linear development.

Key Features of Git:

- Distributed Development: Each developer has a local copy of the entire development history, ensuring work can continue even if a server is offline.
- Data Integrity: Git uses a data model that ensures the cryptographic integrity of every part of the project.
- Branching and Merging: Git offers powerful tools for branching and merging, facilitating various workflows and making it easy to experiment and revert changes.
- Speed: Git is designed for performance, with operations like branching, merging, and committing being incredibly fast.
- Open Source: Git is an open-source project, available freely for everyone.

Basic Git Commands:

- `git init`: Initializes a new Git repository.
- `git clone [url]`: Creates a local copy of a remote repository.
- `git add [file]`: Adds a file to the staging area.
- `git commit -m "[commit message]"`: Commits the staged changes with a message.
- `git push [alias] [branch]`: Pushes committed changes to a remote repository.
- `git pull [alias] [branch]`: Fetches changes from the remote repository and merges them into the current branch.
- `git branch`: Lists, creates, or deletes branches.
- `git checkout [branch]`: Switches to a specified branch.

Using Git with Jenkins

Integrating Git with Jenkins enables automation of the build, test, and deployment phases of your project's lifecycle each time a change is made to your Git repository. This integration is a cornerstone of Continuous Integration/Continuous Deployment (CI/CD) practices.

Setting up Git in Jenkins:

1. Install Git on the Jenkins Server: Ensure Git is installed on the Jenkins server so that Jenkins can interact with Git repositories.

2. Install the Git Plugin in Jenkins: The Git plugin is essential for integrating Git repositories with Jenkins jobs.
   - Navigate to "Manage Jenkins" > "Manage Plugins" > "Available" tab.
   - Search for "Git plugin" and install it.

3. Configure Git in Jenkins:
   - Go to "Manage Jenkins" > "Global Tool Configuration".
   - Scroll down to the "Git" section and ensure the path to the Git executable is correctly set. If Jenkins is on the same server where Git is installed, the default settings usually work fine.

Creating a Jenkins Job for a Git Repository:

1. Create a New Job: Select "New Item" on the Jenkins dashboard, enter a name for your job, and choose a job type (e.g., Freestyle project).

2. Configure Source Code Management:
   - In the job configuration, find the "Source Code Management" section.
   - Select "Git" and enter the repository URL.
   - If your repository is private, you'll also need to configure credentials. Jenkins supports various credential types, including username/password and SSH keys.

3. Configure Build Triggers:
   - Choose how you want Jenkins to be notified of changes to the Git repository. Common options include polling the repository or using webhooks for real-time notifications.

4. Add Build Steps:
   - Configure the actions Jenkins should take when changes are detected, such as executing shell scripts or Maven goals.

5. Save and Run the Job: Save your configuration and run the job to test the integration.

Advanced Integration:

- Webhooks: For more efficient integration, configure a webhook in your Git repository hosting service (like GitHub, GitLab, or Bitbucket) to notify Jenkins of changes automatically.
- Pipeline as Code: For more complex workflows, consider defining your build, test, and deployment steps in a Jenkinsfile using the Pipeline plugin. This approach allows you to version control your CI/CD process alongside your application code.

Integrating Git with Jenkins automates the process of building and testing your software, making it easier to maintain code quality and accelerate the development cycle. By leveraging Jenkins' powerful automation alongside Git's robust version control capabilities, teams can achieve efficient and reliable CI/CD workflows.

14. Application

Application we used is Fake News Detection

This Application needs a CSV file where all the articles are stored .From that csv file this application tries to detect which articles are fake and which are real articles.
The proliferation of fake news has emerged as a significant challenge in the digital age, undermining public discourse, manipulating opinions, and eroding trust in media. The need for automated tools to detect and mitigate the spread of false information is more critical than ever. This project outlines the development of a Python-based program designed to automatically detect fake news articles contained within CSV (Comma-Separated Values) files. The program leverages natural language processing (NLP) techniques, machine learning algorithms, and data analysis methodologies to analyze and classify news articles as "fake" or "real."

Key Components of the Program:

1. Data Preprocessing: The initial phase involves cleaning and preparing the data for analysis. This step includes removing irrelevant features, handling missing values, and normalizing the text data by removing punctuation, stopwords, and applying stemming or lemmatization techniques.

2. Feature Extraction: Utilizing NLP techniques, the program transforms the preprocessed text data into a format that machine learning algorithms can work with. Techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings are employed to convert text into numerical vectors, capturing the significance of words and phrases within the articles.

3. Model Selection and Training: Various machine learning models, including Naive Bayes, Support Vector Machines (SVM), and deep learning models like Recurrent Neural Networks (RNNs), are evaluated for their effectiveness in classifying news articles. The models are trained on a labeled dataset, where each article is tagged as "fake" or "real," allowing the algorithm to learn the distinguishing features of fake news.

4. Evaluation and Validation: The program assesses the performance of the models using metrics such as accuracy, precision, recall, and F1 score. Cross-validation techniques ensure that the models are robust and generalize well to unseen data.

5. Deployment: The final model is integrated into the program, providing a user-friendly interface that allows users to input CSV files containing news articles. The program processes the file, applies the trained model, and outputs a classification for each article along with a confidence score.

Challenges and Solutions:

- Variability of Fake News: Fake news articles can vary widely in style, topic, and fabrication level. To address this, the program utilizes a diverse and comprehensive dataset for training, ensuring the model can generalize across different types of fake news.
- Bias and Overfitting: Machine learning models might become biased or overfit to the training data. The program employs techniques such as regularization, dropout (in neural networks), and balanced training sets to mitigate these issues.
- Real-time Detection: Processing large volumes of data in real-time can be computationally intensive. The program optimizes algorithms for efficiency and can be scaled using cloud computing resources for high-volume processing.

This Python program represents a significant step towards automating the detection of fake news articles, providing a tool that can assist journalists, fact-checkers, and social media platforms in identifying and curbing the spread of misinformation. By combining advanced NLP techniques with machine learning, the program offers a scalable and effective solution to one of the most pressing issues of the digital age.

15. Deploying Application using Jenkins

Deployed application using script in a Jenkins Pipeline, written in Groovy syntax, designed to automate two primary tasks: fetching the latest code from a Git repository and running a script in a virtual environment on a remote server. This pipeline is configured to execute on any available agent configured in Jenkins, which means it's not restricted to running on a specific node. Let's break down the pipeline stages for a clearer understanding:

Stage 1: `fetch-code`

In this stage, the pipeline executes a shell command to connect to a remote server (`192.168.144.213`) via SSH as the `root` user. Once logged in, it navigates to the directory `/nfs/slurm-rpms/HPC-SLURM/` and executes a `git pull` command. This step is intended to update the local codebase with the latest changes from the corresponding Git repository, ensuring that the most current version of the code is used for subsequent steps.

Key Points:
- SSH Connection: It uses SSH for remote server access. This requires the Jenkins server to have SSH keys configured for passwordless login to the remote server.
- Git Pull: Ensures the remote server's working directory has the latest code.

Stage 2: `Run-code`

After updating the code, the pipeline moves on to executing the code in a specific environment. This stage involves several steps:

1. Create a Virtual Environment: It connects to the same remote server and installs `virtualenv` using `pip3` if it's not already installed. Then, it creates a new virtual environment in the directory `/nfs/slurm-rpms/` named `venv`.
2. Activate Virtual Environment: Before running the script, it activates the newly created virtual environment by sourcing the `venv/bin/activate` script. This step ensures that any Python packages are installed and run within this isolated environment.
3. Install Dependencies: With the virtual environment activated, it installs several Python packages (`numpy`, `pandas`, `sklearn`) that are presumably required to run the script successfully.
4. Submit a Job to SLURM: Finally, it submits a job to SLURM (a highly scalable cluster management and job scheduling system) using the `sbatch` command to run `myscript1.sh`. This script likely contains commands to execute a Python script or another program that utilizes the installed packages and updated code.

Key Points:

- Virtual Environment: Isolates the Python environment to avoid conflicts between project dependencies.
- SLURM Job Submission: Utilizes SLURM for job scheduling on the HPC cluster, indicating that the tasks require significant computational resources or need to be managed in a cluster environment.

Considerations and Best Practices

- Security: Using `root` for SSH connections and operations is generally discouraged due to security implications. Consider using a non-root user with the necessary permissions.
- SSH Key Management: Jenkins should securely manage SSH keys, possibly using the Credentials Plugin to handle authentication details.
- Error Handling: The script should include error handling, especially for remote operations, to ensure the pipeline can gracefully manage failures or unexpected outcomes.
- Environmental Variables: For improved maintainability, consider using environmental variables for server addresses, paths, and other configuration details.

This Jenkins Pipeline automates the process of updating and running a codebase on a remote HPC cluster, demonstrating how Jenkins can orchestrate complex workflows involving code updates, environment setup, dependency management, and job scheduling in a high-performance computing context.

19. Code

```
XCAT FINAL SCRIPT>>>>>


# !/bin/bash
# To declare static Array
#arr=(1 2)


# loops iterate through a
# set of values until the
# list (arr) is exhausted


yum -y install yum-utils

wget -P /etc/yum.repos.d https://xcat.org/files/xcat/repos/yum/latest/xcat-core/xcat-
core.repo

wget -P /etc/yum.repos.d https://xcat.org/files/xcat/repos/yum/xcat-
dep/rh7/x86_64/xcat-dep.repo

yum -y install xCAT

. /etc/profile.d/xcat.sh


ifconfig ens34 192.168.1.1 netmask 255.255.255.0 up

chdef -t site dhcpinterfaces="xcatmn|ens34"

copycds /root/CentOS-7-x86_64-DVD-1908.iso

lsdef -t osimage

#### NODE BOOTING WITH FOR LOOP ###########

#!/bin/bash

# Arrays of node names, IP addresses, and MAC addresses
nodes=("hpc001" "hpc002" "hpc003")
ips=("192.168.1.2" "192.168.1.3" "192.168.1.4")
macs=("52:54:00:00:00:01" "52:54:00:00:00:02" "52:54:00:00:00:03")

# Length of arrays
length=${#nodes[@]}

# Loop through the arrays
for ((i=0; i<$length; i++))
do
    node="${nodes[$i]}"
    ip="${ips[$i]}"
    mac="${macs[$i]}"

    # Check if node definition already exists
    if xcatdef=$(lsdef -t node "$node" 2>/dev/null); then
        echo "Node definition for $node already exists. Skipping."
    Else
```

```
        # Create node definition using mkdef
        mkdef -t node "$node"
        echo "Node definition for $node created."
    fi
    # Assign IP address and MAC address to the node
    chdef -t node "$node" ip="$ip" mac="$mac"
   chtab key=system passwd.username=root passwd.password=root
    echo "Assigned IP address $ip and MAC address $mac to $node."
done

makehosts
makenetworks
makedhcp -n
makedhcp -a
makedns -n
makedns -a

nodeset compute osimage=centos7.7-x86_64-install-compute
```

```
JENKINS MACHINE SETUP ...

1) Jenkins PC

----->
       intall jenkins as a service ...
https://www.linuxbuzz.com/install-jenkins-on-rhel-rockylinux-almalinux/#google_vignette

------>install docker in rocky
https://docs.rockylinux.org/gemstones/docker/

->cat /etc/passwd
       jenkins if bin/bash false ....
   then ,,, sudo usermod -s /bin/bash jenkins
--> su - jenkins
---->chown -R  jenkins /demo
--> su - jenkins
ssh production@192.168.144.213
if not,,,,then

---ssh-keygen -t rsa
       ---> ssh-copy-id root@192.168.144.213
              ----->now ssh root@192.168.144.213

open browser localhost:8080 and paste below code for jenkins pipeline.....
```

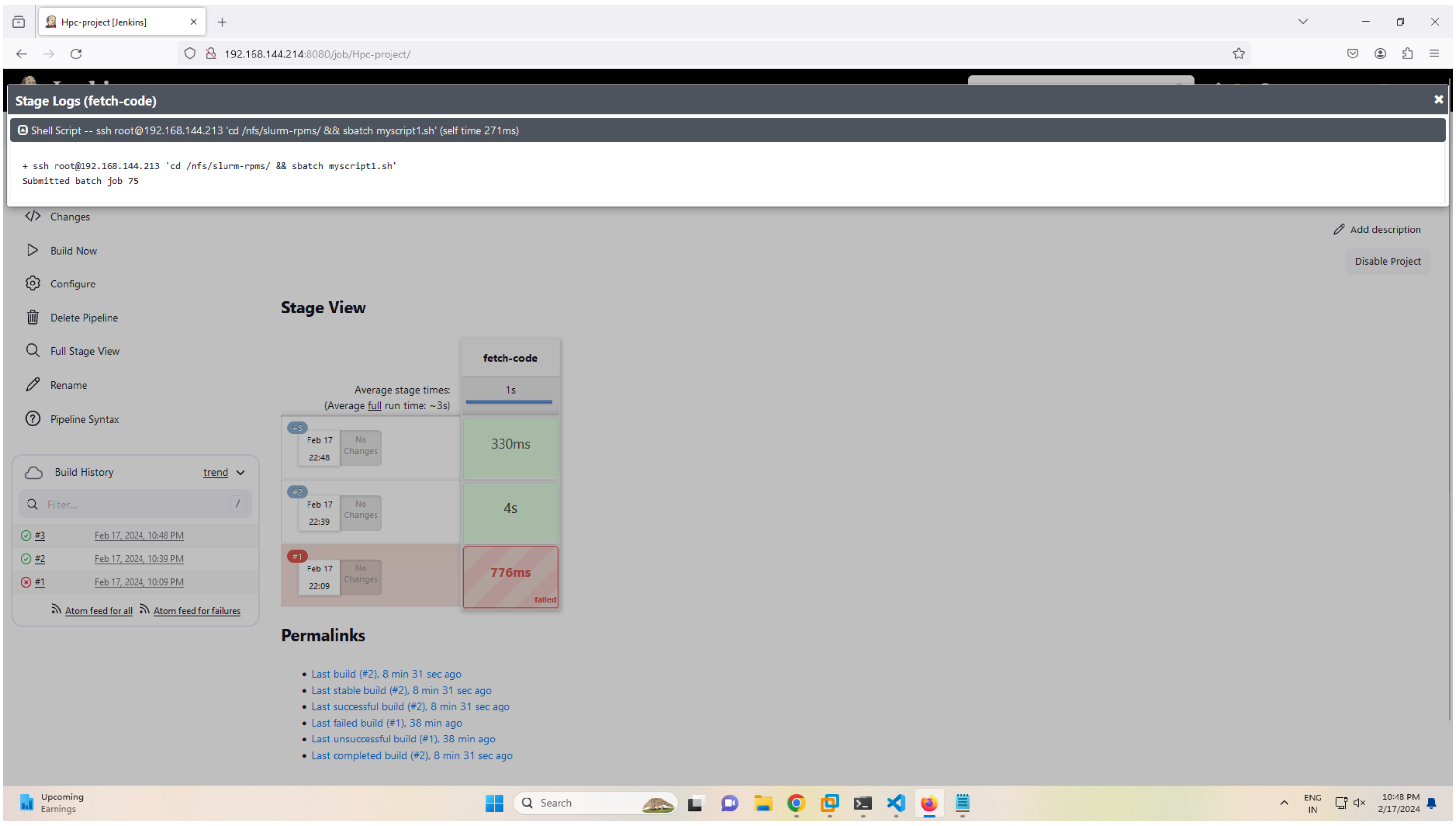
```
SLURM INSTALLATION SETUP>>>


REPOSITORY ....

https://github.com/Sandip1717/HPC-SLURM.git/
```

JENKINS PIPELINE SCRIPT >>>>>

```
pipeline{
    agent any

    stages{

        stage('fetch-code'){
            steps{
                sh '''
                ssh root@192.168.144.213 'cd /nfs/slurm-rpms/HPC-SLURM/ && git pull'
                '''
            }
        }

        stage('Run-code'){
            steps{
                sh '''


                ssh root@192.168.144.213 'pip3 install virtualenv && cd /nfs/slurm-
rpms/ && virtualenv venv && source venv/bin/activate && pip3 install numpy pandas
sklearn && sbatch myscript1.sh'
                '''
            }
        }
    }
}
```

20. Result

- **Pipeline**



Stage Logs (fetch-code)

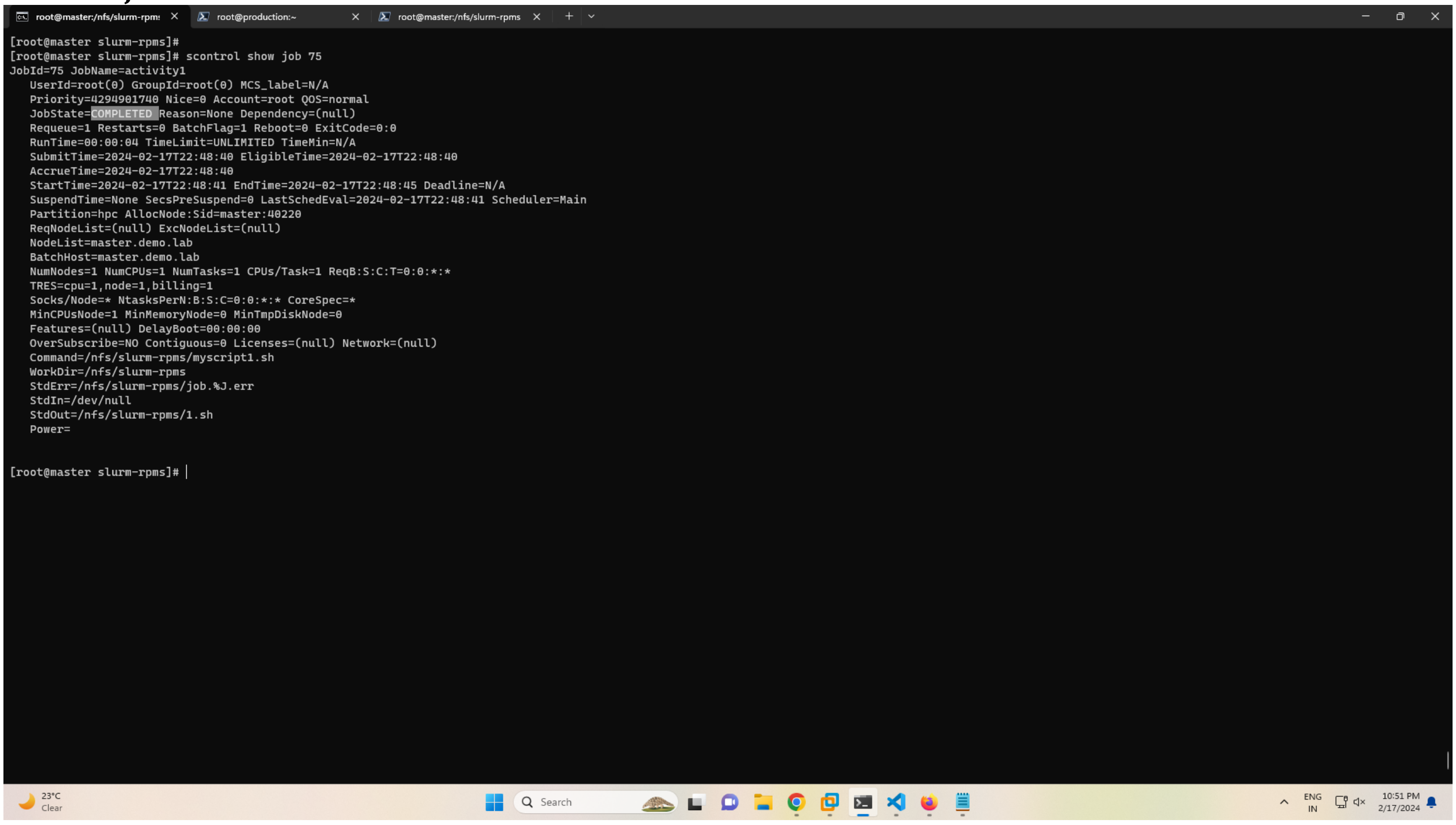
Shell Script -- ssh root@192.168.144.213 'cd /nfs/slurm-rpms/ && sbatch myscript1.sh' (self time 271ms)

```
+ ssh root@192.168.144.213 'cd /nfs/slurm-rpms/ && sbatch myscript1.sh'
Submitted batch job 75
```

Hpc-project [Jenkins]
192.168.144.214:8080/job/Hpc-project/
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Rename
Pipeline Syntax
Build History
trend
#3 Feb 17, 2024, 10:48 PM
#2 Feb 17, 2024, 10:39 PM
#1 Feb 17, 2024, 10:09 PM
Atom feed for all
Atom feed for failures
Add description
Disable Project
Stage View
fetch-code
Average stage times:
(Average full run time: ~3s)
1s
#3 Feb 17 22:48 No Changes 330ms
#2 Feb 17 22:39 No Changes 4s
#1 Feb 17 22:09 No Changes 776ms failed
Permalinks

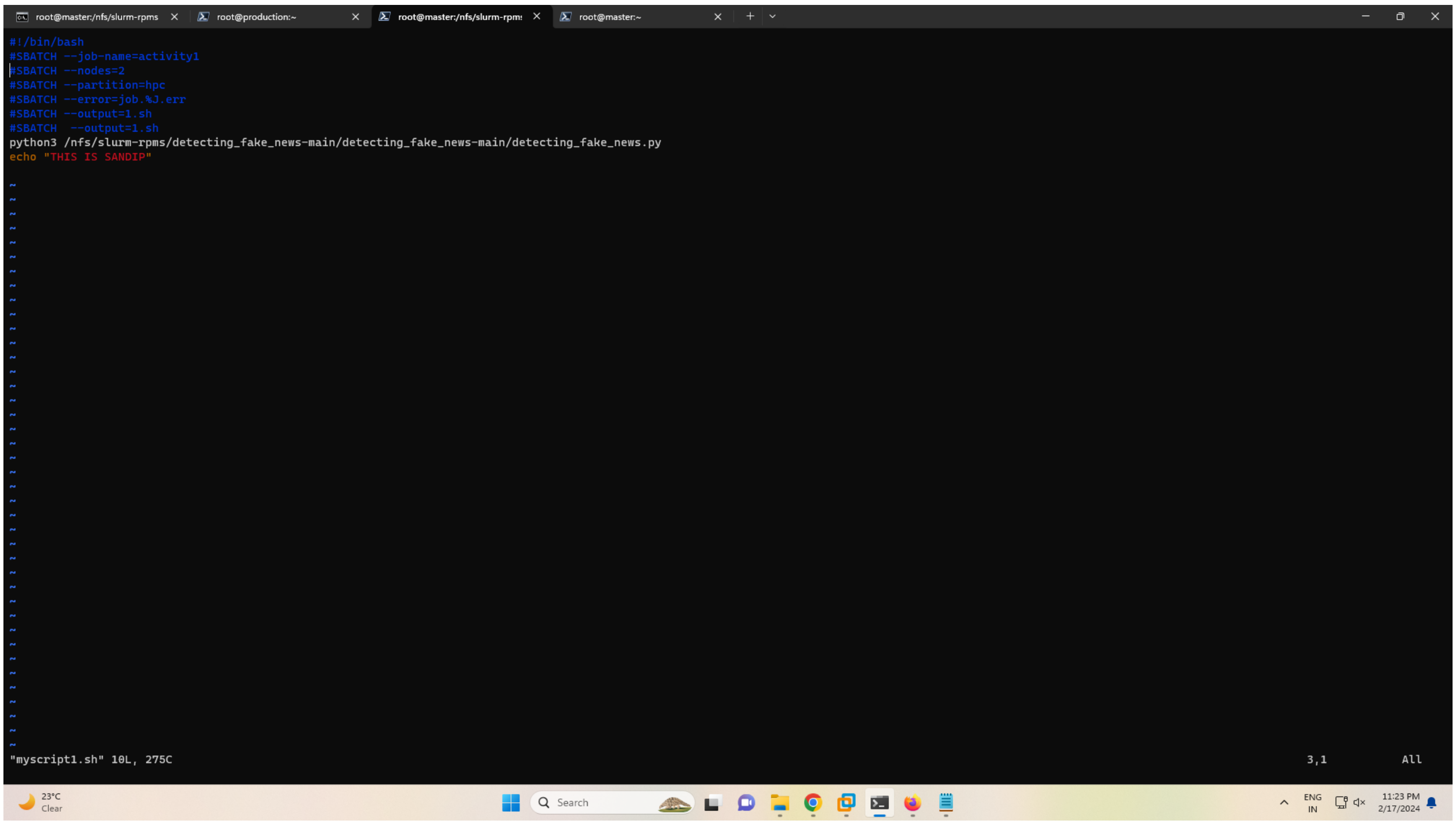
- Last build (#2), 8 min 31 sec ago
- Last stable build (#2), 8 min 31 sec ago
- Last successful build (#2), 8 min 31 sec ago
- Last failed build (#1), 38 min ago
- Last unsuccessful build (#1), 38 min ago
- Last completed build (#2), 8 min 31 sec ago

- **Job Executed**



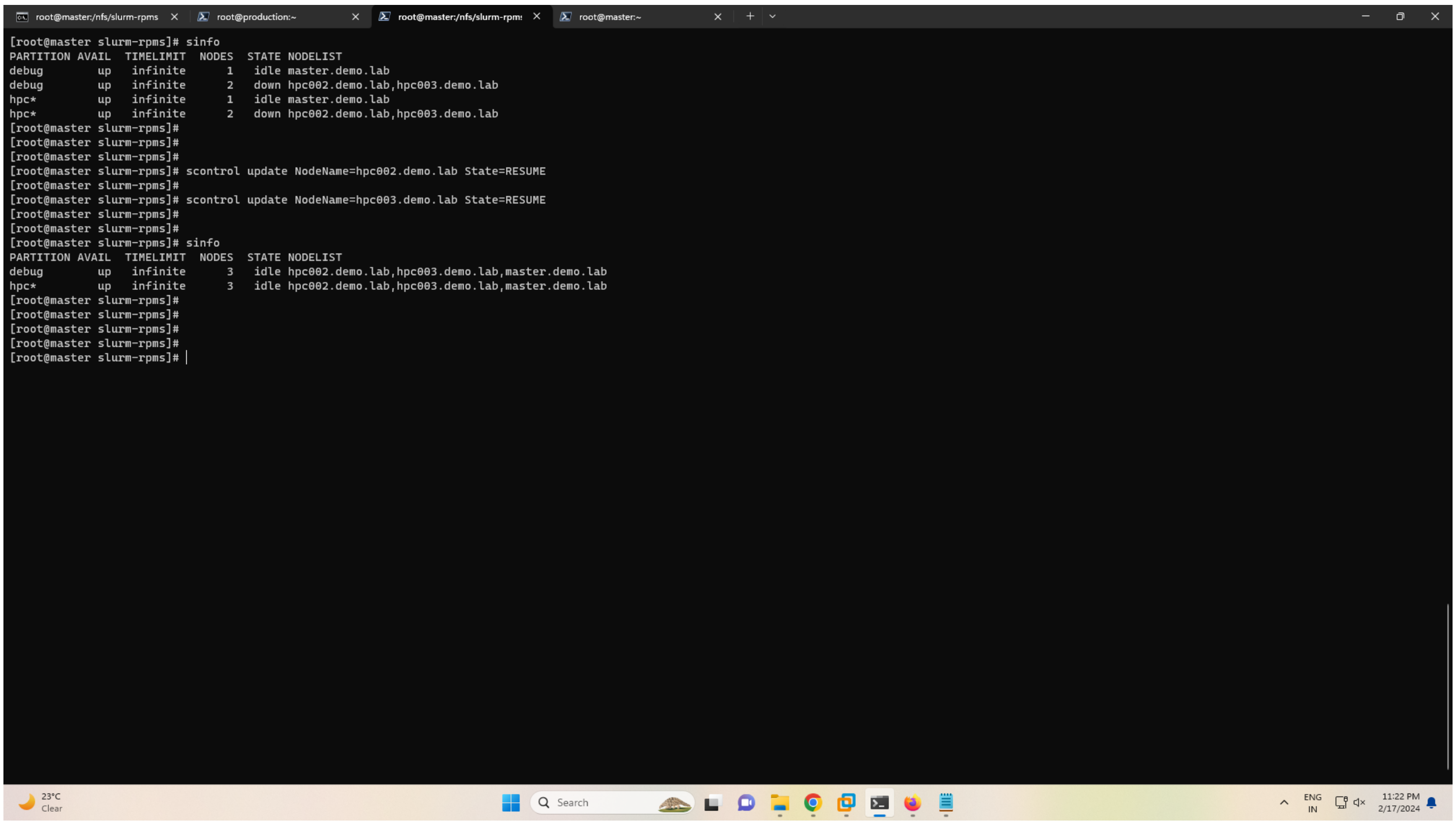
```
[root@master slurm-rpms]#
[root@master slurm-rpms]# scontrol show job 75
JobId=75 JobName=activity1
   UserId=root(0) GroupId=root(0) MCS_label=N/A
   Priority=4294901740 Nice=0 Account=root QOS=normal
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:04 TimeLimit=UNLIMITED TimeMin=N/A
   SubmitTime=2024-02-17T22:48:40 EligibleTime=2024-02-17T22:48:40
   AccrueTime=2024-02-17T22:48:40
   StartTime=2024-02-17T22:48:41 EndTime=2024-02-17T22:48:45 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2024-02-17T22:48:41 Scheduler=Main
   Partition=hpc AllocNode:Sid=master:40220
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=master.demo.lab
   BatchHost=master.demo.lab
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,node=1,billing=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
   Command=/nfs/slurm-rpms/myscript1.sh
   WorkDir=/nfs/slurm-rpms
   StdErr=/nfs/slurm-rpms/job.%J.err
   StdIn=/dev/null
   StdOut=/nfs/slurm-rpms/1.sh
   Power=


[root@master slurm-rpms]#
```

- **Slurm Job Script**



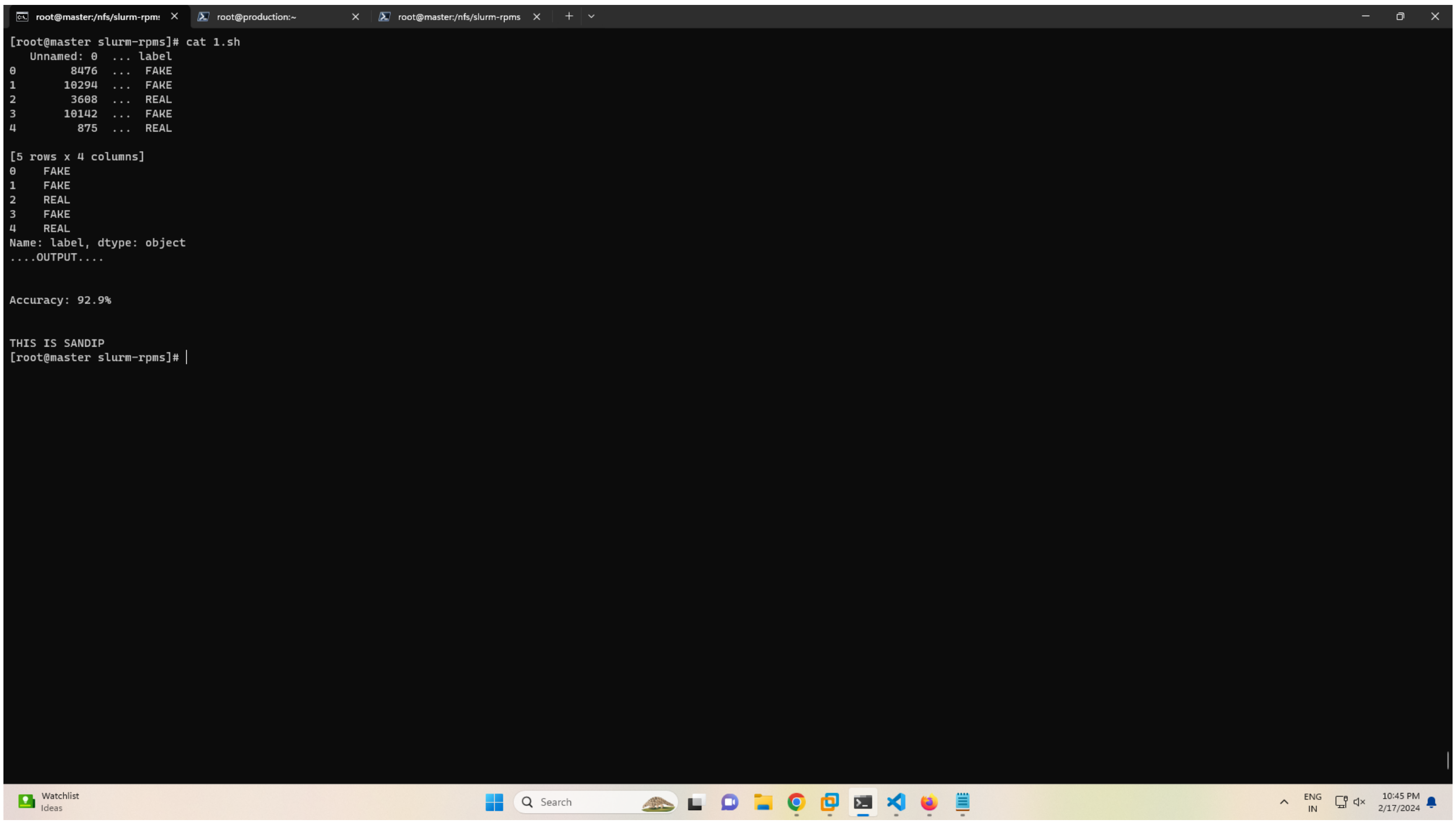
```
#!/bin/bash
#SBATCH --job-name=activity1
#SBATCH --nodes=2
#SBATCH --partition=hpc
#SBATCH --error=job.%J.err
#SBATCH --output=1.sh
#SBATCH  --output=1.sh
python3 /nfs/slurm-rpms/detecting_fake_news-main/detecting_fake_news-main/detecting_fake_news.py
echo "THIS IS SANDIP"

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"myscript1.sh" 10L, 275C                                                                          3,1           All
```

- **Slurm Node List**



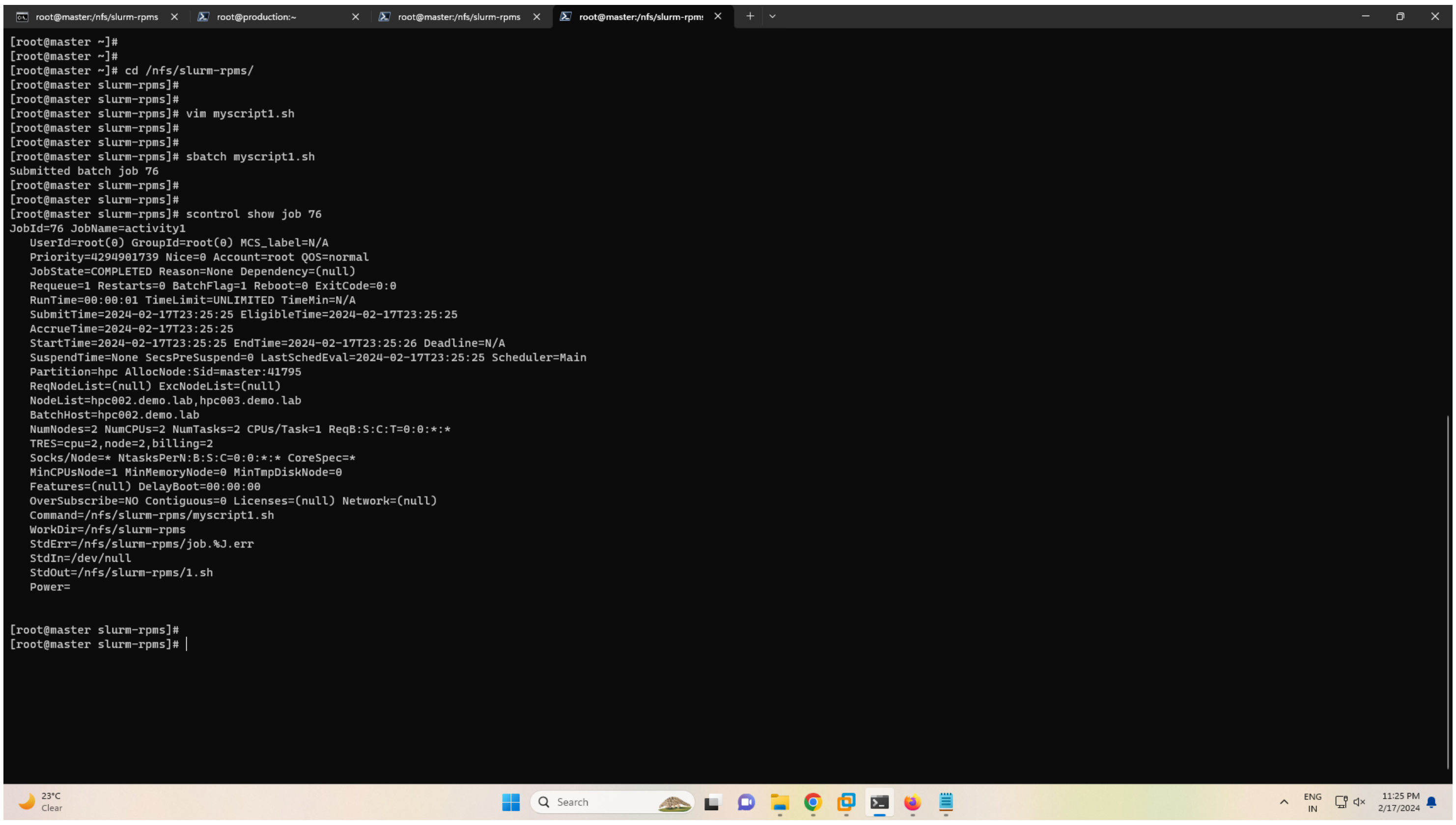
```
[root@master slurm-rpms]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up   infinite      1   idle master.demo.lab
debug        up   infinite      2   down hpc002.demo.lab,hpc003.demo.lab
hpc*         up   infinite      1   idle master.demo.lab
hpc*         up   infinite      2   down hpc002.demo.lab,hpc003.demo.lab
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]# scontrol update NodeName=hpc002.demo.lab State=RESUME
[root@master slurm-rpms]#
[root@master slurm-rpms]# scontrol update NodeName=hpc003.demo.lab State=RESUME
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up   infinite      3   idle hpc002.demo.lab,hpc003.demo.lab,master.demo.lab
hpc*         up   infinite      3   idle hpc002.demo.lab,hpc003.demo.lab,master.demo.lab
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]# |
```

- **Job Output at Slurm Machine**



```
[root@master slurm-rpms]# cat 1.sh
   Unnamed: 0  ... label
0        8476  ...  FAKE
1       10294  ...  FAKE
2        3608  ...  REAL
3       10142  ...  FAKE
4         875  ...  REAL

[5 rows x 4 columns]
0    FAKE
1    FAKE
2    REAL
3    FAKE
4    REAL
Name: label, dtype: object
....OUTPUT....


Accuracy: 92.9%


THIS IS SANDIP
[root@master slurm-rpms]# |
```

- **Nodes on job**



```
[root@master ~]#
[root@master ~]#
[root@master ~]# cd /nfs/slurm-rpms/
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]# vim myscript1.sh
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]# sbatch myscript1.sh
Submitted batch job 76
[root@master slurm-rpms]#
[root@master slurm-rpms]#
[root@master slurm-rpms]# scontrol show job 76
JobId=76 JobName=activity1
   UserId=root(0) GroupId=root(0) MCS_label=N/A
   Priority=4294901739 Nice=0 Account=root QOS=normal
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:01 TimeLimit=UNLIMITED TimeMin=N/A
   SubmitTime=2024-02-17T23:25:25 EligibleTime=2024-02-17T23:25:25
   AccrueTime=2024-02-17T23:25:25
   StartTime=2024-02-17T23:25:25 EndTime=2024-02-17T23:25:26 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2024-02-17T23:25:25 Scheduler=Main
   Partition=hpc AllocNode:Sid=master:41795
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=hpc002.demo.lab,hpc003.demo.lab
   BatchHost=hpc002.demo.lab
   NumNodes=2 NumCPUs=2 NumTasks=2 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=2,node=2,billing=2
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
   Command=/nfs/slurm-rpms/myscript1.sh
   WorkDir=/nfs/slurm-rpms
   StdErr=/nfs/slurm-rpms/job.%J.err
   StdIn=/dev/null
   StdOut=/nfs/slurm-rpms/1.sh
   Power=


[root@master slurm-rpms]#
[root@master slurm-rpms]# |
```

21. Conclusion

The successful execution of implementing a CI/CD pipeline to deploy applications on an HPC cluster represents a significant achievement in the field of DevSecOps, integrating software development (Dev), security (Sec), and operations (Ops) into a unified process. This project entailed setting up a Jenkins machine to automate the deployment process on an HPC cluster, which was constructed using xCAT for node provisioning and managed with Slurm for resource scheduling. The entire process underscores the importance of planning, designing, developing, testing, deploying, and operating phases in the SSDLC (Secure Software Development Life Cycle) within a DevSecOps framework. Here's a conclusion highlighting the key accomplishments and insights gained:

Key Accomplishments:

- Infrastructure Setup: The project successfully established a robust infrastructure comprising a Jenkins machine (Production Machine), an HPC Master for node provisioning and routing (using xCAT and iptables), and compute nodes for task execution. The HPC Master also served as a gateway for compute nodes, facilitating network access and resource management via Slurm.

- Integration and Automation: Through Jenkins, automation was achieved from code retrieval to application deployment on the HPC cluster. This automation included SSH configuration for seamless communication between Jenkins, the HPC Master, and compute nodes, and NFS setup for common storage, ensuring consistency across the environment.

- Resource Management and Scheduling: The implementation of Slurm on the HPC Master and compute nodes enabled efficient resource management and job scheduling, proving crucial for optimizing the utilization of compute resources within the HPC cluster.

- Security and Accessibility: The project addressed security concerns by configuring iptables for routing and network access control, alongside SSH for secure remote operations. These measures ensured a secure and manageable infrastructure, aligning with DevSecOps principles.

Insights Gained:

- Collaboration between Tools: The integration of Jenkins, xCAT, Slurm, iptables, and NFS showcased the power of leveraging multiple tools in harmony to build a sophisticated CI/CD pipeline. Each tool played a vital role in the infrastructure, from automation and provisioning to resource management and network security.

- DevSecOps as a Holistic Approach: The project exemplified the DevSecOps approach by integrating security considerations throughout the development lifecycle, from initial planning to deployment and operation. This holistic approach ensures that security is not an afterthought but a fundamental aspect of the infrastructure.

- Scalability and Flexibility: The setup demonstrated scalability, with the potential to add more compute nodes as needed, and flexibility in deploying various applications. The CI/CD pipeline can be adapted to different projects, showcasing the versatility of the infrastructure.

- Continuous Improvement: The operation phase underlined the importance of continuous monitoring and improvement, essential for maintaining the efficiency and security of the CI/CD pipeline and the HPC environment.

By meticulously following the steps outlined in the SSDLC within a DevSecOps framework, this project successfully implemented a CI/CD infrastructure for deploying applications on an HPC cluster. The integration of Jenkins for automation, xCAT for node provisioning, iptables for routing, NFS for common storage, and Slurm for resource management exemplifies a comprehensive approach to modern software deployment on HPC systems. This achievement not only enhances operational efficiency but also reinforces the importance of security and scalability in software development and deployment processes.

22. References

1 "Implementing DevSecOps: Security in the DevOps Lifecycle" by Yogesh Singh and Prabath Siriwardena, Apress, 2018.

2. "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations" by Gene Kim, Jez Humble, Patrick Debois, and John Willis, IT Revolution Press, 2016.

3. "Continuous Security in DevOps" by Paula Thrasher, IEEE Security & Privacy, vol. 16, no. 5, pp. 68-71, 2018.

4. Deploying app on HPC - https://dl.acm.org/doi/pdf/10.1145/3219104.3219147

5. xCat –
https://openhpc.community/downloads/

https://github.com/openhpc/ohpc/wiki/1.3.X

6. Jenkins –

https://azdigi.com/blog/en/linux-server-en/tools-en/how-to-install-jenkins-on-centos-7/

https://www.jenkins.io/doc/book/installing/linux/

7. Slurm - https://github.com/Artlands/Install-Slurm