KodeKloud 100 Days DevOps Challenge

Table of Contents

Linux

Day 01: Linux User Setup with Non-Interactive Shell

Task Description:

To accommodate the backup agent tool's specifications, the system admin team at xFusionCorp Industries requires the creation of a user with a non-interactive shell.

Here's your task:
Create a user named john with a non-interactive shell on App Server 2.

Solution:

To create a Linux user with a non-interactive shell (**a shell that prevents the user from logging in interactively**), we can assign them a restricted shell like ***/usr/sbin/nologin*** or ***/bin/false***

Why use **nologin** instead of **false**?

- nologin shows a message like "**This account is currently not available**" when attempting login.
- false silently exits with no message (less user-friendly).

First connect remotely to the App Server 2 by, ***Password: Am3ric@***

```
ssh steve@172.16.238.11
```

Then check user exists or not

```
id john
```

If not then create by,

```
sudo adduser john --shell /sbin/nologin
or
sudo adduser john --shell /bin/false
```

Then check user by

```
sudo cat /etc/passwd | grep john
```

Use Case:

Many services (e.g., `nginx`, `mysql`, `www-data`) run under specific users. These users only need to **own files** or **run processes**, not log in. Non-interactive shell ensures **security** by preventing unintended logins.

Sometimes applications need **dedicated users** for accessing databases, queues, or APIs. These users only need credentials to authenticate services, not log into the system.

Day 02: Temporary User Setup with Expiry

Task Description:

As part of the temporary assignment to the Nautilus project, a developer named james requires access for a limited duration. To ensure smooth access management, a temporary user account with an expiry date is needed.
Here's what you need to do:
Create a user named james on App Server 3 in Stratos Datacenter. Set the expiry date to 2024-04-15, ensuring the user is created in lowercase as per standard protocol.

Solution:

Creating a temporary user with an expiry date is used in several real-world scenarios, especially in DevOps, sysadmin, and security-focused environments.

First connect remotely to the App Server 3 by, ***Password: BigGr33n***

```
ssh banner@172.16.238.12
```

Then check user exists or not

```
id james
```

If not then create by,

```
sudo adduser james -e 2024-04-15
or
sudo adduser james --expiredate 2024-04-15
```

Verify the expiry date

```
sudo chage -l james
```

Then check user by

```
sudo cat /etc/passwd | grep james
```

Use Case:

- When developers, contractors, or interns need access only for a specific project duration
- Reduces risk of **orphaned accounts** (accounts that remain active after a user leaves).
- Meets **compliance standards** (like ISO, SOC 2, HIPAA) where time-bound access is mandatory.
- Supports **principle of least privilege** with time-limited access.

Day 03: Secure Root SSH Access

Task Description:

Following security audits, the xFusionCorp Industries security team has rolled out new protocols, including the restriction of direct root SSH login.
Your task is to disable direct SSH root login on all app servers within the Stratos Datacenter.

Solution:

Disabling **direct root SSH login** is a **best security practice** in system administration.

Login to each app server:

```
ssh tony@172.16.238.10
```

Switch to superuser (if not already):

```
sudo -i
```

Edit the SSH configuration file:

```
vi /etc/ssh/sshd_config
```

Find and update the line:

```
PermitRootLogin no
```

If the line is commented (#PermitRootLogin yes), uncomment it and change yes to no.

Restart the SSH service:

```
systemctl restart sshd
```

Repeat on All App Servers (stapp01 , stapp02 , stapp03)

Use case

- Hackers often target the `root` account via brute-force login attempts. Disabling root login eliminates this common target.
- Users must log in as a regular user and then elevate privileges (`sudo`) only when necessary. This provides **better control** over who can perform administrative actions.
- The `root` account has full control — accidental use can lead to catastrophic mistakes. Requiring `sudo` adds a layer of confirmation before critical commands are run.

Day 04: Secure Root SSH Access

Task Description:

In a bid to automate backup processes, the xFusionCorp Industries sysadmin team has developed a new bash script named xfusioncorp.sh. While the script has been distributed to all necessary servers, it lacks executable permissions on App Server 3 within the Stratos Datacenter.

Your task is to grant executable permissions to the /tmp/xfusioncorp.sh script on App Server 3. Additionally, ensure that all users have the capability to execute it.

Solution:

First connect remotely to the App Server 3 by, ***Password: BigGr33n***

```
ssh banner@172.16.238.12
```

Then Check permission by,

```
ls -ltr /tmp/xfusioncorp.sh
```

Check permission of the file byThen change permission by,

```
sudo chmod o+rx /tmp/xfusioncorp.sh
```

Day 05: SElinux Installation and Configuration

Task Description:

Following a security audit, the xFusionCorp Industries security team has opted to enhance application and server security with SELinux. To initiate testing, the following requirements have been established for App server 1 in the Stratos Datacenter:

- Install the required SELinux packages.
- Permanently disable SELinux for the time being; it will be re-enabled after necessary configuration changes.
- No need to reboot the server, as a scheduled maintenance reboot is already planned for tonight.
- Disregard the current status of SELinux via the command line; the final status after the reboot should be disabled.

Solution:

SELinux (**Security-Enhanced Linux**) is a **security feature** built into the Linux kernel that enforces mandatory access control (**MAC**) policies.
Think of it as an extra security guard on top of the normal Linux permissions system.

What SELinux adds
SELinux introduces **security policies** that control:

- **Which processes** can access **which files**.
- **What actions** processes can take.
- **Which network ports** services can use.

It works by labeling every:

- **File**
- **Directory**
- **Process**
- **Network port**
  …with a **security context**, and then enforcing rules about which labels can interact.

Modes of SELinux

1. **Enforcing** – Rules are applied and violations are blocked (most secure).
2. **Permissive** – Rules are checked, violations are only logged (for troubleshooting).
3. **Disabled** – SELinux is turned off.

First connect remotely to the App Server 1 by, ***Password: Ir0nM@n***

```
ssh banner@172.16.238.10
```

Install SELinux packages

```
sudo yum install selinux* -y
```

Permanent SELinux settings are controlled by ***/etc/selinux/config***.
Edit the config file:

```
sudo vi /etc/selinux/config
```

Change the line from

```
SELINUX=enforcing
```

to:

```
SELINUX=disabled
```

Save and exit.

Do NOT reboot now, Since a maintenance reboot is scheduled for tonight, you don't need to reboot now.

Verification

```
sestatus
```

Use Case:

The main **use case of SELinux** is to **limit the damage** if an application, process, or user account is compromised — especially in **servers and critical systems**.

SELinux is useful when you want **extra defense** against both external hackers and insider threats, ensuring that even if normal permissions fail, an attacker still faces a locked-down system.

Day 06: Create a Cron Job

Task Description:

The `Nautilus` system admins team has prepared scripts to automate several day-to-day tasks. They want them to be deployed on all app servers in `Stratos DC` on a set schedule. Before that they need to test similar functionality with a sample cron job. Therefore, perform the steps below:

a. Install `cronie` package on all `Nautilus` app servers and start `crond` service.
b. Add a cron `*/5 * * * * echo hello > /tmp/cron_text` for `root` user.

Solution:

On **each** Nautilus App Server, SSH log in as root (or use `sudo`) and run:

switch user as root

```
sudo -i
```

Install cronie

```
yum install -y cronie
```

Enable and start the crond service

```
systemctl enable crond
systemctl start crond
systemctl status crond
```

On some systems `cronie` might already be installed; in that case the `yum` command will just confirm it.

Edit root's crontab:

```
crontab -e
```

Add this line:

```
*/5 * * * * echo hello > /tmp/cron_text
```

Save and exit

Day 07: Linux SSH Authentication

Task Description:

The system admins team of `xFusionCorp Industries` has set up some scripts on `jump host` that run on regular intervals and perform operations on all app servers in `Stratos Datacenter`. To make these scripts work properly we need to make sure the `thor` user on jump host has password-less SSH access to all app servers through their respective sudo users (i.e `tony` for app server 1). Based on the requirements, perform the following:
Set up a password-less authentication from user `thor` on jump host to all app servers through their respective sudo users

Solution:

Switch to thor user on Jump Host

```
sudo su - thor
```

Generate SSH key (on Jump Host as thor) If thor doesn't already have an SSH key:

```
ssh-keygen -t rsa
```

Press Enter for all prompts (no passphrase for password-less).

Copy public key to each App Server's sudo user:

```
ssh-copy-id <sudo_user>@<app_server_ip_or_hostname>
```

Example for App Server 1:

```
ssh-copy-id tony@<app_server1_ip>
```

Repeat for each server
You'll enter the sudo user's password **once** here (to copy the key). After that, no password is needed for SSH.
Test password-less SSH

```
ssh tony@<app_server1_ip>
```

It should log in without asking for a password.

Use case:
If scripts had to prompt for a password for every connection, automation would break (no human to type passwords during a cron run).

Example Scenario

```
ssh tony@app-server1 "sudo systemctl restart httpd"
```

Without password-less SSH, each line would stop and wait for a password.

Day 08: Install Ansible

Task Description:

During the weekly meeting, the Nautilus DevOps team discussed about the automation and configuration management solutions that they want to implement. While considering several options, the team has decided to go with `Ansible` for now due to its simple setup and minimal pre-requisites. The team wanted to start testing using Ansible, so they have decided to use `jump host` as an Ansible controller to test different kind of tasks on rest of the servers.

Install `ansible` version `4.10.0` on `Jump host` using `pip3` only. Make sure Ansible binary is available globally on this system, i.e all users on this system are able to run Ansible commands.

Solution:

Update package index and install pip3 (if missing)

```
sudo apt update
sudo apt install python3-pip -y
```

Install Ansible 4.10.0 with pip3 system-wide

```
sudo pip3 install ansible==4.10.0
```

Verify installation

```
ansible --version
```

Ensure it's in PATH for all users
/usr/local/bin should already be in the PATH. You can check with:

```
echo $PATH
```

If not, add this line to `/etc/profile`:

```
export PATH=/usr/local/bin:$PATH
```

Day 09: MariaDB Troubleshooting

Task Description:

There is a critical issue going on with the Nautilus application in Stratos DC. The production support team identified that the application is unable to connect to the database. After digging into the issue, the team found that mariadb service is down on the database server.
Look into the issue and fix the same.

Solution:

Connect to the Database Server

```
ssh peter@172.16.239.10
```

Check MariaDB Service Status

```
sudo systemctl status mariadb
```

Investigate Logs for Errors

```
sudo journalctl -u mariadb --no-pager -n 20
# or check the error log file
sudo cat /var/log/mariadb/mariadb.log
```

Issue Summary

- **Problem**: Nautilus application unable to connect to the database.
- **Root Cause**: MariaDB service (mariadb) was down due to a failed initialization process.
- **Error Observed**:

```
mariadb-prepare-db-dir[3220]: Make sure the /var/lib/mysql is empty before
running mariadb-prepare-db-dir.
mariadb.service: Control process exited, code=exited, status=1/FAILURE
```

Troubleshooting Steps Taken
Verified ***/var/lib/mysql*** Contents

```
sudo ls -la /var/lib/mysql
```

Observation:
The directory contained residual files from a previous database instance.
No critical production data was present (confirmed with the team).

Stopped MariaDB (if partially running)

```
sudo systemctl stop mariadb
```

Cleared Old Database Files

```
sudo rm -rf /var/lib/mysql/*
```

Reinitialized the Database

```
sudo mariadb-install-db --user=mysql --basedir=/usr
--datadir=/var/lib/mysql
```

Fixed Permissions

```
sudo chown -R mysql:mysql /var/lib/mysql
sudo chmod 700 /var/lib/mysql
```

Started MariaDB Service

```
sudo systemctl start mariadb
```

Verified Service Status

```
sudo systemctl status mariadb
```

Resolution Summary

- Root Cause: MariaDB failed to start because /var/lib/mysql was not empty.
- Action Taken:
    - Removed residual files from /var/lib/mysql.
    - Reinitialized the database.
    - Corrected permissions.
    - Restarted MariaDB successfully.
- Verification:
    - Service is now active (running).
    - Nautilus application can now connect to the database.

Day 10: Linux Bash Scripts

Task Description:

The production support team of xFusionCorp Industries is working on developing some bash scripts to automate different day to day tasks. One is to create a bash script for taking websites backup. They have a static website running on App Server 2 in Stratos Datacenter, and they need to create a bash script named media_backup.sh which should accomplish the following tasks. (Also remember to place the script under /scripts directory on App Server 2).

- Create a zip archive named xfusioncorp_media.zip of /var/www/html/media directory.
- Save the archive in /backup/ on App Server 2. This is a temporary storage, as backups from this location will be clean on weekly basis. Therefore, we also need to save this backup archive on Nautilus Backup Server.
- Copy the created archive to Nautilus Backup Server server in /backup/ location.
- Please make sure script won't ask for password while copying the archive file. Additionally, the respective server user (for example, tony in case of App Server 1) must be able to run it.

Solution:

SSH into App Server 2

```
ssh steve@172.16.238.11
```

Create the media_backup.sh script in the /scripts directory

```
cd /scripts
touch media_backup.sh
chmod +x media_backup.sh
```

Open the file and vi editor via

```
vi media_backup.sh
```

add the following content

```
#!/bin/bash

echo "Create a zip archive"
zip -r xfusioncorp_media.zip /var/www/html/media

echo "Save the archive in backup directory on app server 2"
cp /scripts/xfusioncorp_media.zip /backup/

echo "Secure copy the zip file in backup server"
```

```
scp -r /backup/xfusioncorp_media.zip clint@172.16.238.16:/backup/
```

Before executing this script **Set Up SSH Key-Based Authentication** connection between **app server 2** user and **backup server user**

Generate SSH Key Pair on App Server 2:

```
ssh-keygen -t rsa
```

Press Enter to accept default locations and leave the passphrase empty if you want password-less authentication.

Copy the Public Key to Nautilus Backup Server:

```
ssh-copy-id clint@172.16.238.16
```

Test SSH Login:

```
ssh clint@172.16.238.16
```

Ensure you can log in without a password.

Day 11: Install and Configure Tomcat Server

Task Description:

The Nautilus application development team recently finished the beta version of one of their Java-based applications, which they are planning to deploy on one of the app servers in Stratos DC. After an internal team meeting, they have decided to use the tomcat application server. Based on the requirements mentioned below complete the task:

- Install tomcat server on App Server 3.
- Configure it to run on port 5004.
- There is a ROOT.war file on Jump host at location /tmp.
- Deploy it on this tomcat server and make sure the webpage works directly on base URL i.e curl http://stapp03:5004

Solution:

Apache Tomcat is an open-source web server and Servlet container for Java code. It's a production-ready Java development tool used to implement many types of Jakarta EE (formerly known as Java EE) specifications.
Tomcat is considered a web server instead of an application server because it functions as a web server and Servlet container. It doesn't provide the full feature set from Jakarta EE, but that isn't necessarily a disadvantage.

Connect to App Server 3

```
ssh banner@stapp03
```

Install Tomcat

```
sudo yum install -y tomcat
```

Change Tomcat Listening Port
Edit the ***server.xml*** file

```
sudo vi /etc/tomcat/server.xml
```

Locate:

```
<Connector port="8080" protocol="HTTP/1.1"
```

Change to:

```
<Connector port="5004" protocol="HTTP/1
```

Enable and Start Tomcat Service

```
sudo systemctl enable tomcat
```

```
sudo systemctl start tomcat
sudo systemctl status tomcat
```

Make sure that jump host user **thor** have ssh access to the **app server 3**
Transfer the WAR File from Jump Host

```
scp /tmp/ROOT.war banner@stapp03:/tmp
```

Run this scp command on Jump Host

Deploy **ROOT.war**

```
sudo cp /tmp/ROOT.war /usr/share/tomcat/webapps/
```

Run this copy command on app server 3

Restart Tomcat

```
sudo systemctl restart tomcat
```

Verify Deployment

```
curl http://stapp03:5004
```

Expected: HTML output from deployed application.

Day 12: Linux Network Services

Task Description:

Our monitoring tool has reported an issue in Stratos Datacenter. One of our app servers has an issue, as its Apache service is not reachable on port 5003 (which is the Apache port). The service itself could be down, the firewall could be at fault, or something else could be causing the issue.

Use tools like telnet, netstat, etc. to find and fix the issue. Also make sure Apache is reachable from the jump host without compromising any security settings.

Once fixed, you can test the same using command curl http://stapp01:5003 command from jump host.

Solution:

First hit the apache server from Jump Host

```
curl http://stapp01:5003
```

The server is not reachable from Jump Host

SSH to the app server 1

```
ssh tony@stapp01  //Ir0nM@n
```

App server is CentOS machine so check httpd service

Verified Apache service status

```
sudo systemctl status httpd
```

Error found:

```
(98)Address already in use: AH00072: could not bind to address 0.0.0.0:5003
```

Checked port usage

```
sudo netstat -tulnp | grep 5003
```

Output showed:

```
127.0.0.1:5003 ... LISTEN ... sendmail
```

Port 5003 was already bound by sendmail.

Stopped and disabled sendmail

```
sudo systemctl stop sendmail
sudo systemctl disable sendmail
```

Restarted Apache

```
sudo systemctl start httpd
sudo systemctl enable httpd
```

The https service is up and running but still not reachable from Jump Host.

Check Firewall

Firewalld and UFW are not installed on the app server so it does not blockage.

Checked iptables rules

```
sudo iptables -L -n -v
```

Allowed traffic on port 5003:

```
sudo iptables -I INPUT -p tcp --dport 5003 -j ACCEPT
sudo service iptables save
```

Validation
On stapp01:

```
curl http://localhost:5003
```

From Jump Host:

```
curl http://stapp01:5003
```

Day 13: IPtables Installation And Configuration

Task Description:

We have one of our websites up and running on our `Nautilus` infrastructure in `Stratos DC`. Our security team has raised a concern that right now Apache's port i.e `8089` is open for all since there is no firewall installed on these hosts. So we have decided to add some security layer for these hosts and after discussions and recommendations we have come up with the following requirements:

- Install `iptables` and all its dependencies on each app host.
- Block incoming port `8089` on all apps for everyone except for LBR host.
- Make sure the rules remain, even after system reboot.

Solution:

IPtables:

iptables is a built-in firewall in Linux. It works at the kernel level using the Netfilter framework. It decides whether to allow, block, or redirect network packets based on a set of rules. Rules are organized into chains (**INPUT**, **OUTPUT**, **FORWARD**) inside tables (most common: **filter**)

Think of it like a security checkpoint: every packet entering or leaving the server must pass through iptables, where rules decide what happens to it.

Install iptables

```
sudo yum install -y iptables iptables-services
```

Add Rules
LBR host IP is 172.16.238.14
Run the following on each App Host:

```
# Allow traffic from LBR host on port 8089
sudo iptables -A INPUT -p tcp -s 172.16.238.14 --dport 8089 -j ACCEPT

# Block everyone else on port 8089
sudo iptables -A INPUT -p tcp --dport 8089 -j DROP
```

Save Rules (Persistence)
To make rules survive reboot:

```
sudo service iptables save
sudo systemctl enable iptables
```

Rules will be saved in /etc/sysconfig/iptables.

Verify Rules
List rules:

```
sudo iptables -L -n -v
```

You should see something like:

```
ACCEPT  tcp  --  192.168.1.100   0.0.0.0/0   tcp dpt:8089
DROP    tcp  --  0.0.0.0/0       0.0.0.0/0   tcp dpt:8089
ACCEPT  tcp  --  0.0.0.0/0       0.0.0.0/0   tcp dpt:22
```

Day 14: Linux Process Troubleshooting

Task Description:

The Task is to check the apache service is running on **all app servers**, identify which app server has a problem on apache service. Make sure apache service is running on Port **3004**

Solution:

First check which app server have problem by curl command,

```
curl http://stapp01:3004
curl http://stapp02:3004
curl http://stapp03:3004
```

In my case app server 2 and app server 3 responded and gave the website content, but app server 1 did not respond.

Connect to app server 1

```
ssh tony@stapp01  //Ir0nM@n
```

Then check the httpd service

```
sudo systemctl status httpd
```

Error found:

```
Address already in use
```

Check which service is running on 3004

```
sudo netstat -tulnp | grep 3004
```

sendmail service is running on 3004, then stop and disable the sendmail service

```
sudo systemctl stop sendmail
sudo systemctl disable sendmail
```

After disbale the sendmail service, start the httpd service

```
sudo systemctl start httpd
sudo systemctl enable httpd
sudo systemctl status httpd
```

Exit the app server 1 and hit the app server 1 from the Jump Host by curl command

```
curl http://stapp01:3004
```

Day 15: Setup SSL for Nginx

Task Description:

The system admins team of `xFusionCorp Industries` needs to deploy a new application on `App Server 1` in `Stratos Datacenter`. They have some pre-requites to get ready that server for application deployment. Prepare the server as per requirements shared below:

- Install and configure `nginx` on `App Server 1`
- On `App Server 1` there is a self signed SSL certificate and key present at location `/tmp/nautilus.crt` and `/tmp/nautilus.key`. Move them to some appropriate location and deploy the same in Nginx.
- Create an `index.html` file with content `Welcome!` under Nginx document root.
- For final testing try to access the `App Server 1` link (either hostname or IP) from `jump host` using curl command. For example `curl -Ik https://<app-server-ip>/`

Solution:

Connect to remote app server 1

```
ssh tony@stapp01  // Ir0nM@n
```

Install Nginx on app server 1
On CentOS, Nginx is not always in the default repos, so install **EPEL (Extra Packages for Enterprise Linux)** first:

```
sudo yum install epel-release -y
sudo yum install -y nginx
```

Enable and start nginx:

```
sudo systemctl start nginx
sudo systemctl enable nginx
sudo systemctl status nginx
```

Move the SSL Certificate and Key on ***/etc/ssl*** directory

```
sudo mv /tmp/nautilus.crt /etc/ssl
sudo mv /tmp/nautilus.key /etc/ssl
```

Change the Key permissions

```
cd /etc/ssl
sudo chmod 600 nautilus.key
```

Create ***index.html*** file in ***/var/www/html***

```
mkdir -p /var/www/html
echo "Welcome!" | sudo tee /var/www/html/index.html
```

Configure Nginx
Create nginx file in ***/etc/nginx/conf.d/default.conf***

```
sudo vi /etc/nginx/conf.d/default.conf
```

Edit the file

```
sudo vi /etc/nginx/conf.d/default.conf
```

```
server {
    listen 443 ssl;
    server_name _;

    ssl_certificate     /etc/ssl/nautilus.crt;
    ssl_certificate_key /etc/ssl/nautilus.key;

    root /var/www/html;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
server {
    listen 80;
    return 301 https://$host$request_uri;
}
```

Test config:

```
sudo nginx -t
sudo systemctl reload nginx
```

Test from jump host
From jump host, run:

```
curl -Ik https://stapp01
```

Expected output:

```
HTTP/1.1 200 OK
...
```

Day 16: Install and Configure Nginx as an LBR

Task Description:

Day by day traffic is increasing on one of the websites managed by the `Nautilus` production support team. Therefore, the team has observed a degradation in website performance. Following discussions about this issue, the team has decided to deploy this application on a high availability stack i.e on `Nautilus` infra in `Stratos DC`. They started the migration last month and it is almost done, as only the LBR server configuration is pending. Configure LBR server as per the information given below:

- Install `nginx` on `LBR` (load balancer) server.
- Configure load-balancing with the an `http` context making use of all `App Servers`. Ensure that you update only the main Nginx configuration file located at `/etc/nginx/nginx.conf`.
- Make sure you do not update the apache port that is already defined in the apache configuration on all app servers, also make sure apache service is up and running on all app servers.
- Once done, you can access the website using `StaticApp` button on the top bar.

Solution:

Connect remotely to Load Balancer server

```
ssh loki@stlb01  // Mischi3f
```

Install Nginx on LBR (Load Balancer)Server

```
sudo yum install -y nginx
sudo systemctl enable nginx
sudo systemctl start nginx
```

Apache Configuration on App Servers

Check Apache is running in all app servers (**stapp01**, **stapp02**, **stapp03**)
ssh in each app server and check the apache service status by,

```
sudo systemctl status httpd

# Start If Stop
sudo systemctl start httpd
```

Check listening port:

```
cat /etc/httpd/conf/httpd.conf | grep Listen
```

Output:

```
Listen 8083
```

Each App Server (stapp01, stapp02, stapp03) is running Apache on port 8083. This port must not be changed.

Configure Nginx Load Balancer

We are required to modify only the main configuration file:

```
 sudo vi /etc/nginx/nginx.conf
```

Update the file as follows:

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    upstream backend {
        server stapp01:8083;
        server stapp02:8083;
        server stapp03:8083;
    }
    server {
        listen 80;

        location / {
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Validate Configuration
Check Nginx syntax:

```
sudo nginx -t
```

Reload Nginx:

```
sudo systemctl reload nginx
```

Verification

1. Open the StaticApp button in the Nautilus portal (or access the LBR server's IP in a browser).
2. Nginx will round-robin requests to all three app servers.

To confirm via CLI, run multiple requests:

```
curl http://172.16.238.14
curl http://172.16.238.14
curl http://172.16.238.14
```

3. You should see responses alternating between stapp01, stapp02, and stapp03

Day 17: Install and Configure PostgreSQL

Task Description:

The `Nautilus` application development team has shared that they are planning to deploy one newly developed application on `Nautilus` infra in `Stratos DC`. The application uses PostgreSQL database, so as a pre-requisite we need to set up PostgreSQL database server as per requirements shared below:
PostgreSQL database server is already installed on the `Nautilus` database server.

- Create a database user `kodekloud_rin` and set its password to `LQfKeWWxWD`.
- Create a database `kodekloud_db4` and grant full permissions to user `kodekloud_rin` on this database.

`Note:` Please do not try to restart PostgreSQL server service.

Solution:

Connect remotely to the database server

```
ssh peter@stdb01  // Sp!dy
```

Checl postgresql service

```
sudo systemctl status postgresql
```

Steps Performed
Switch to the PostgreSQL system user

```
sudo -i -u postgres
```

Connect to PostgreSQL shell

```
psql
```

Created user kodekloud_rin with the required password.

```
CREATE USER kodekloud_rin WITH PASSWORD 'LQfKeWWxWD';
```

Create a database

```
CREATE DATABASE kodekloud_db4;
```

Grant permissions

```
GRANT ALL PRIVILEGES ON DATABASE kodekloud_db4 TO kodekloud_rin;
```

Verification

```
\du    // Check roles
\l     // Check Databases
\q     // exit postgresql shell
```

Day 18: Configure LAMP Server

Task Description:

xFusionCorp Industries is planning to host a `WordPress` website on their infra in `Stratos Datacenter`. They have already done infrastructure configuration—for example, on the storage server they already have a shared directory `/vaw/www/html` that is mounted on each app host under `/var/www/html` directory. Please perform the following steps to accomplish the task:

- Install httpd, php and its dependencies on all app hosts.
- Apache should serve on port `3000` within the apps.
- Install/Configure `MariaDB server` on DB Server.
- Create a database named `kodekloud_db1` and create a database user named `kodekloud_rin` identified as password `B4zNgHA7Ya`. Further make sure this newly created user is able to perform all operation on the database you created.
- Finally you should be able to access the website on LBR link, by clicking on the `App` button on the top bar. You should see a message like `App is able to connect to the database using user kodekloud_gem`

Solution:

Install **Apache**, **PHP**, and dependencies (on all App servers)
SSH into each App Server (**stapp01**, **stapp02**, **stapp03**)

```
sudo yum install -y httpd php php-mysqlnd php-fpm php-cli
```

Enable Apache

```
sudo systemctl enable httpd
```

Configure Apache to serve on port **3000**
Edit Apache config file:

```
sudo vi /etc/httpd/conf/httpd.conf
```

Find the line:

```
Listen 80
```

and change it to:

```
Listen 3000
```

Restart Apache:

```
sudo systemctl restart httpd
```

Install/Configure MariaDB on DB Server
SSH into DB Server (**stdb01**)

```
sudo yum install -y mariadb-server
sudo systemctl enable mariadb
sudo systemctl start mariadb
```

Create Database and User
Login to MariaDB:

```
sudo mysql -u root
```

In Mariadb Sheel run:

```
CREATE DATABASE kodekloud_db1;
CREATE USER 'kodekloud_rin'@'%' IDENTIFIED BY 'B4zNgHA7Ya';
GRANT ALL PRIVILEGES ON kodekloud_db1.* TO 'kodekloud_rin'@'%';
FLUSH PRIVILEGES;
EXIT;
```

Also, allow remote connections:

```
sudo vi /etc/my.cnf.d/mariadb-server.cnf
```

Find:

```
[mysqld]
```

and add/modify:

```
bind-address=0.0.0.0
```

Restart MariaDB:

```
sudo systemctl restart mariadb
```

Day 19: Install and Configure Web Application

Task Description:

xFusionCorp Industries is planning to host two static websites on their infra in `Stratos Datacenter`. The development of these websites is still in-progress, but we want to get the servers ready. Please perform the following steps to accomplish the task:

- Install `httpd` package and dependencies on `app server 3`.
- Apache should serve on port `8082`.
- There are two website's backups `/home/thor/official` and `/home/thor/apps` on `jump_host`. Set them up on Apache in a way that `official` should work on the link `http://localhost:8082/official/` and `apps` should work on link `http://localhost:8082/apps/` on the mentioned app server.
- Once configured you should be able to access the website using `curl` command on the respective app server, i.e `curl http://localhost:8082/official/` and `curl http://localhost:8082/apps/`

Solution:

Connect remotely on app server 3

```
ssh banner@stapp03  // BigGr33n
```

Install Apache (**httpd**)

```
sudo yum install -y httpd
```

Configure Apache to run on Port **8082**
Edit Apache config file:

```
sudo vi /etc/httpd/conf/httpd.conf
```

Find the line:

```
Listen 80
```

Change it to:

```
 Listen 8082
```

Start and Enable Apache

```
sudo systemctl enable httpd
sudo systemctl restart httpd
```

Check if Apache is listening on port 8082:

```
sudo ss -tulnp | grep 8082
```

Configure websites (official & apps)

Copy backup websites from **jump host** to **app server 3**:

```
sudo scp -r /home/thor/official banner@stapp03:/var/www/html/
sudo scp -r /home/thor/apps banner@stapp03:/var/www/html/
```

After copy, you should have in app server 3:

```
/var/www/html/official
/var/www/html/apps
```

Create a new Apache config file in app server 3:

```
sudo vi /etc/httpd/conf.d/websites.conf
```

Add the following:

```
 <VirtualHost *:8082>
    DocumentRoot "/var/www/html"
    Alias /official "/var/www/html/official"
    <Directory "/var/www/html/official">
        Require all granted
    </Directory>

    Alias /apps "/var/www/html/apps"
    <Directory "/var/www/html/apps">
        Require all granted
    </Directory>
</VirtualHost>
```

Test with curl in app server 3

```
curl http://localhost:8082/official/
curl http://localhost:8082/apps/
```

Both should return HTML content of the respective sites.

Day 20: Configure Nginx + PHP-FPM Using Unix Sock

Task Description:

The `Nautilus` application development team is planning to launch a new PHP-based application, which they want to deploy on `Nautilus` infra in `Stratos DC`. The development team had a meeting with the production support team and they have shared some requirements regarding the infrastructure. Below are the requirements they shared:

- Install `nginx` on `app server 2` , configure it to use port `8092` and its document root should be `/var/www/html`.
- Install `php-fpm` version `8.2` on `app server 2`, it must use the unix socket `/var/run/php-fpm/default.sock` (create the parent directories if don't exist).
- Configure php-fpm and nginx to work together.
- Once configured correctly, you can test the website using `curl http://stapp02:8092/index.php` command from jump host.

Solution:

Connect remotely on app server 2

```
ssh steve@stapp02  // Am3ric@
```

Install nginx on app server 2

```
sudo yum install -y nginx
```

Enable and start:

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

Install PHP-FPM 8.2 in app server 2

Enable EPEL & Remi repo for EL9

```
sudo dnf install -y epel-release
sudo dnf install -y https://rpms.remirepo.net/enterprise/remi-release-9.rpm
```

Reset default PHP module & enable 8.2

```
sudo dnf module reset php -y
sudo dnf module enable php:remi-8.2 -y
```

Install PHP 8.2 + FPM

```
sudo dnf install -y php php-fpm php-mysqlnd php-cli php-common
```

Verify

```
php-fpm --version
```

Configure PHP-FPM Socket
Edit the pool config:

```
sudo vi /etc/php-fpm.d/www.conf
```

Change:

```
# New socket (per requirement)
listen = /var/run/php-fpm/default.sock

# Set socket permissions
listen.owner = nginx
listen.group = nginx
listen.mode = 0660

# Run php-fpm under nginx user
user = nginx
group = nginx
```

Create parent directory (if missing):

```
sudo mkdir -p /var/run/php-fpm
sudo chown nginx:nginx /var/run/php-fpm
```

Restart php-fpm:

```
sudo systemctl enable php-fpm
sudo systemctl restart php-fpm
```

Configure Nginx
Create new config file:

```
sudo vi /etc/nginx/conf.d/php_app.conf
```

Add the following:

```
server {
    listen 8092;

    root /var/www/html;
    index index.php index.html;

    location / {
```

```
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass unix:/var/run/php-fpm/default.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_index index.php;
    }
}
```

Test and reload:

```
sudo nginx -t
sudo systemctl restart nginx
```

Test from Jump Host

```
curl http://stapp02:8092/index.php
```

Git

Day 21: Set Up Git Repository on Storage Server

Task Description:

The Nautilus development team has provided requirements to the DevOps team for a new application development project, specifically requesting the establishment of a Git repository. Follow the instructions below to create the Git repository on the `Storage server` in the Stratos DC:

- Utilize `yum` to install the `git` package on the `Storage Server`.
- Create a bare repository named `/opt/demo.git` (ensure exact name usage)

Solution:

A **bare repository** is a Git repository without a **working directory**. It only contains the **.git** folder contents: ***objects/***, ***refs/***, ***HEAD***, ***config***, etc. We cannot edit or work on files directly in a bare repo—it's only for storing Git history and sharing with others.

Connect Remotely to the storage server

```
ssh natasha@ststor01  // Bl@kW
```

Install git

```
sudo yum install -y git
```

Create a bare Git repository
Bare repositories are used for central repositories (no working directory).

```
sudo git init --bare /opt/demo.git
```

Verify the repository
Check that the repo exists and is bare:

```
ls -ld /opt/demo.git
```

Inside the directory, you should see files like HEAD, config, objects/, and refs/.

```
ls /opt/demo.git
```

Use Case:

Bare repositories are typically used as central repos for collaboration.
If we want to share a project inside a private network (no cloud), you set up a bare repo and let everyone push/pull.

Day 22: Clone Git Repository on Storage Server

Task Description:

The DevOps team established a new Git repository last week, which remains unused at present. However, the Nautilus application development team now requires a copy of this repository on the `Storage Server` in the Stratos DC. Follow the provided details to clone the repository:

- The repository to be cloned is located at `/opt/news.git`
- Clone this Git repository to the `/usr/src/kodekloudrepos` directory. Ensure no modifications are made to the repository during the cloning process.

Solution:

First Connect Remotely to Storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Navigate to the target directory

```
cd /usr/src/kodekloudrepos
```

Clone the repository
Since the repository is a bare repository (indicated by the .git extension), you can clone it using:

```
git clone /opt/blog.git
```

This will create a directory named blog under ***/usr/src/kodekloudrepos*** containing the cloned repository.

Verify the clone
You can check the contents of the cloned repository to ensure it was cloned correctly:

```
ls -la /usr/src/kodekloudrepos/blog
```

Day 23: Fork a Git Repository

Task Description:

There is a Git server utilized by the Nautilus project teams. Recently, a new developer named Jon joined the team and needs to begin working on a project. To begin, he must fork an existing Git repository. Follow the steps below:

- Click on the `Gitea UI` button located on the top bar to access the Gitea page.
- Login to `Gitea` server using username `jon` and password `Jon_pass123`.
- Once logged in, locate the Git repository named `sarah/story-blog` and `fork` it under the `jon` user.

`Note:` For tasks requiring web UI changes, screenshots are necessary for review purposes. Additionally, consider utilizing screen recording software such as loom.com to record and share your task completion process.

Solution:

This task is done from the Web UI. We are using Gitea for this solution.

Day 24: Git Create Branches

Task Description:

Nautilus developers are actively working on one of the project repositories, `/usr/src/kodekloudrepos/apps`. Recently, they decided to implement some new features in the application, and they want to maintain those new changes in a separate branch. Below are the requirements that have been shared with the DevOps team:

- On `Storage server` in Stratos DC create a new branch `xfusioncorp_apps` from `master` branch in `/usr/src/kodekloudrepos/apps` git repo.
- Please do not try to make any changes in the code

Solution:

Connect Remotely to the Storage server

```
ssh natasha@ststor01 // Bl@kW
```

Go to the repository path

```
cd /usr/src/kodekloudrepos/apps
```

Switch to the root user

```
sudo -i
```

Check the current branch (should be master)

```
git branch
```

If you're not on master, switch

```
git checkout master
```

Create a new branch from master

```
git checkout -b xfusioncorp_apps
```

Verify the new branch exists

```
git branch
```

Day 25: Git Merge Branches

Task Description:

The Nautilus application development team has been working on a project repository `/opt/blog.git`. This repo is cloned at `/usr/src/kodekloudrepos` on `storage server` in `Stratos DC`. They recently shared the following requirements with DevOps team:

Create a new branch `devops` in `/usr/src/kodekloudrepos/blog` repo from `master` and copy the `/tmp/index.html` file (present on `storage server` itself) into the repo. Further, `add/commit` this file in the new branch and merge back that branch into `master` branch. Finally, push the changes to the origin for both of the branches.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Navigate to the repository

```
cd /usr/src/kodekloudrepos/blog
```

Ensure you are on the latest master

```
git checkout master
git pull origin master
```

Create and switch to a new branch devops

```
git checkout -b devops
```

Copy the required file into the repository

```
cp /tmp/index.html .
```

Stage and commit the file in devops branch

```
git add index.html
git commit -m "Added index.html file"
```

Push the devops branch to the remote repository

```
git push origin devops
```

Switch back to the master branch

```
git checkout master
```

Merge the devops branch into master

```
git merge devops
```

Push the updated master branch to the remote repository

```
git push origin master
```

Day 26: Git Manage Remotes

Task Description:

The xFusionCorp development team added updates to the project that is maintained under `/opt/apps.git` repo and cloned under `/usr/src/kodekloudrepos/apps`. Recently some changes were made on Git server that is hosted on `Storage server` in `Stratos DC`. The DevOps team added some new Git remotes, so we need to update remote on `/usr/src/kodekloudrepos/apps` repository as per details mentioned below:

- In `/usr/src/kodekloudrepos/apps` repo add a new remote `dev_apps` and point it to `/opt/xfusioncorp_apps.git` repository.
- There is a file `/tmp/index.html` on same server; copy this file to the repo and add/commit to master branch.
- Finally push `master` branch to this new remote origin.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Add a new remote

```
cd /usr/src/kodekloudrepos/apps
git remote add dev_apps /opt/xfusioncorp_apps.git
```

This will create a new remote named ***dev_apps*** pointing to ***/opt/xfusioncorp_apps.git***.

You can verify:

```
git remote -v
```

You should see ***dev_apps*** listed.

Copy file and commit

```
 cp /tmp/index.html .
```

Add the file to git:

```
 git add .
```

Commit the changes:

```
git commit -m "Add index.html file"
```

Push to new remote

Push the master branch to the new remote:

```
git push dev_apps master
```

Day 27: Git Revert Some Changes

Task Description:

The Nautilus application development team was working on a git repository `/usr/src/kodekloudrepos/cluster` present on `Storage server` in `Stratos DC`. However, they reported an issue with the recent commits being pushed to this repo. They have asked the DevOps team to revert repo HEAD to last commit. Below are more details about the task:

- In `/usr/src/kodekloudrepos/cluster` git repository, revert the latest commit `( HEAD )` to the previous commit (JFYI the previous commit hash should be with `initial commit` message).

- Use `revert cluster` message (please use all small letters for commit message) for the new revert commit.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Go to the repo directory

```
cd /usr/src/kodekloudrepos/cluster
```

Check the commit history

```
git log --oneline
```

Revert the latest commit (HEAD)

```
git revert HEAD -m 1
```

(it will open the editor)

Set the commit message. If it opens the editor, replace everything with:

```
revert cluster
```

Save and close the editor.
Verify

```
git log --oneline
```

Day 28: Git Cherry Pick

Task Description:

The Nautilus application development team has been working on a project repository `/opt/news.git`. This repo is cloned at `/usr/src/kodekloudrepos` on `storage server` in `Stratos DC`. They recently shared the following requirements with the DevOps team:

There are two branches in this repository, `master` and `feature`. One of the developers is working on the `feature` branch and their work is still in progress, however they want to merge one of the commits from the `feature` branch to the `master` branch, the message for the commit that needs to be merged into `master` is `Update info.txt`. Accomplish this task for them, also remember to push your changes eventually.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Go to the cloned repo

```
cd /usr/src/kodekloudrepos/news
```

Check the branches

```
git branch
```

You should see:
* master
  feature

Switch to the feature branch and find the commit

```
git checkout feature
git log --oneline
```

Look through the logs and note the commit hash for the commit with the message: Update info.txt

Switch back to the master branch

```
git checkout master
```

Cherry-pick the commit

```
git cherry-pick a1b2c3d
```

(Replace a1b2c3d with the actual commit hash you found earlier.)

Push the changes

```
git push origin master
```

Day 29: Manage Git Pull Requests

Task Description:

`Max` want to push some new changes to one of the repositories but we don't want people to push directly to `master` branch, since that would be the final version of the code. It should always only have content that has been reviewed and approved. We cannot just allow everyone to directly push to the master branch. So, let's do it the right way as discussed below:

SSH into `storage server` using user `max`, password `Max_pass123` . There you can find an already cloned repo under `Max` user's home.

Max has written his story about The 🦊 Fox and Grapes 🍇

Max has already pushed his story to remote git repository hosted on `Gitea` branch `story/fox-and-grapes`

Check the contents of the cloned repository. Confirm that you can see Sarah's story and history of commits by running `git log` and validate author info, commit message etc.

Max has pushed his story, but his story is still not in the `master` branch. Let's create a Pull Request(PR) to merge Max's `story/fox-and-grapes` branch into the `master` branch

Click on the `Gitea UI` button on the top bar. You should be able to access the `Gitea` page.

- UI login info:
- Username: `max`
- Password: `Max_pass123`
- PR title : `Added fox-and-grapes story`
- PR pull from branch: `story/fox-and-grapes` (source)
- PR merge into branch: `master` (destination)

Before we can add our story to the `master` branch, it has to be reviewed. So, let's ask `tom` to review our PR by assigning him as a reviewer
Add tom as reviewer through the Git Portal UI

- Go to the newly created PR
- Click on Reviewers on the right
- Add tom as a reviewer to the PR

Now let's review and approve the PR as user `Tom`

Login to the portal with the user `tom`
Logout of `Git Portal UI` if logged in as `max`

UI login info:

- Username: `tom`
- Password: `Tom_pass123`
- PR title : `Added fox-and-grapes story`
- Review and merge it.

Great stuff!! The story has been merged! 👏
`Note:` For these kind of scenarios requiring changes to be done in a web UI, please take screenshots so that you can share it with us for review in case your task is marked incomplete. You may also consider using a screen recording software such as loom.com to record and share your work.

Solution:

Connect to the storage server

```
ssh max@ststor01  //Max_pass123
```

Go to max home

```
cd /home/max
```

Check the repository

```
ls
```

We see ***story-blog*** repository in the max home directory

All the Stuff are done through the WebUI of Gitea

Day 30: Git Hard Reset

Task Description:

The Nautilus application development team was working on a git repository `/usr/src/kodekloudrepos/blog` present on `Storage server` in `Stratos DC`. This was just a test repository and one of the developers just pushed a couple of changes for testing, but now they want to clean this repository along with the commit history/work tree, so they want to point back the `HEAD` and the branch itself to a commit with message `add data.txt file`. Find below more details:

- In `/usr/src/kodekloudrepos/blog` git repository, reset the git commit history so that there are only two commits in the commit history i.e `initial commit` and `add data.txt file`.
- Also make sure to push your changes.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Go to the repo

```
cd /usr/src/kodekloudrepos/blog
```

Check commit history

```
git log --oneline
```

Look for the commit with message:
add data.txt file
Note down its commit hash (let's call it HASH).

Reset the branch to that commit
We want to rebuild the history so only 2 commits remain.

```
git reset --hard <HASH>
```

Check log again

```
git log -oneline
```

Force push to remote

```
git push origin master -f
```

Day 31: Git Stash

Task Description:

The Nautilus application development team was working on a git repository `/usr/src/kodekloudrepos/apps` present on `Storage server` in `Stratos DC`. One of the developers stashed some in-progress changes in this repository, but now they want to restore some of the stashed changes. Find below more details to accomplish this task:

Look for the stashed changes under `/usr/src/kodekloudrepos/apps` git repository, and restore the stash with `stash@{1}` identifier. Further, commit and push your changes to the origin.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Go to the repo

```
cd /usr/src/kodekloudrepos/apps
```

Check stash list

```
git stash list
```

Apply the stash@{1}

`git stash apply stash@{1}`

Check the changes

```
git status
```

Stage the changes

```
git add .
```

Commit the changes

```
git commit -m "Restored changes from stash@{1}"
```

Push to origin

```
git push origin master
```

Day 32: Git Rebase

Task Description:

The Nautilus application development team has been working on a project repository `/opt/demo.git`. This repo is cloned at `/usr/src/kodekloudrepos` on `storage server` in `Stratos DC`. They recently shared the following requirements with DevOps team:

One of the developers is working on `feature` branch and their work is still in progress, however there are some changes which have been pushed into the `master` branch, the developer now wants to `rebase` the `feature` branch with the `master` branch without loosing any data from the `feature` branch, also they don't want to add any `merge commit` by simply merging the `master` branch into the `feature` branch. Accomplish this task as per requirements mentioned.
Also remember to push your changes once done.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Go to the repo

```
cd /usr/src/kodekloudrepos/demo
```

Check branches (just to verify you're on the right branch)

```
git branch
```

Switch to feature branch (where developer is working)

```
git checkout feature
```

Fetch latest changes from remote (to ensure master is up to date)

```
git fetch origin
```

Rebase feature branch with master

```
git rebase origin/master
```

This replays the commits of the feature on top of the updated master.

Verify commit history

```
git log --oneline --graph --decorate --all
```

You should see feature commits sitting on top of master, without a merge commit.

Push the rebased branch
Since rebase rewrites history, you'll need to force push:

```
git push origin feature --force
```

Day 33: Resolve Git Merge Conflicts

Task Description:

Sarah and Max were working on writting some stories which they have pushed to the repository. Max has recently added some new changes and is trying to push them to the repository but he is facing some issues. Below you can find more details:

SSH into `storage server` using user `max` and password `Max_pass123`. Under `/home/max` you will find the `story-blog` repository. Try to push the changes to the origin repo and fix the issues. The `story-index.txt` must have titles for all 4 stories. Additionally, there is a typo in `The Lion and the Mooose` line where `Mooose` should be `Mouse`.

Click on the `Gitea UI` button on the top bar. You should be able to access the `Gitea` page. You can login to `Gitea` server from UI using username `sarah` and password `Sarah_pass123` or username `max` and password `Max_pass123`.

`Note:` For these kind of scenarios requiring changes to be done in a web UI, please take screenshots so that you can share it with us for review in case your task is marked incomplete. You may also consider using a screen recording software such as loom.com to record and share your work.

Solution:

SSH into storage server

```
ssh max@ststor01  //Max_pass123
```

Navigate to repository

```
cd /home/max/story-blog
```

Check repo status:

```
git status
```

Pull latest changes from origin

```
git pull origin master
```

Resolve Merge Conflicts in story-index.txt file

```
sudo vi story-index.txt
```

Make sure it lists all 4 stories with their titles. Correct typo mistake
After resolving, mark as resolved:

```
git add .
git commit -m "Resolve merge Conflicts"
```

Push changes

```
git push origin master
```

Verify in Gitea UI
Go to Gitea UI (top bar button).
Login using either:
    sarah / Sarah_pass123
    max / Max_pass123
Open story-blog repo → confirm:
    story-index.txt has all 4 story titles.
    Typo fixed (Mouse).
    Latest commit is visible.

Day 34: Git Hook

Task Description:

The Nautilus application development team was working on a git repository `/opt/official.git` which is cloned under `/usr/src/kodekloudrepos` directory present on `Storage server` in `Stratos DC`. The team want to setup a hook on this repository, please find below more details:

- Merge the `feature` branch into the `master` branch`, but before pushing your changes complete below point.
- Create a `post-update` hook in this git repository so that whenever any changes are pushed to the `master` branch, it creates a release tag with name `release-2023-06-15`, where `2023-06-15` is supposed to be the current date. For example if today is `20th June, 2023` then the release tag must be `release-2023-06-20`. Make sure you test the hook at least once and create a release tag for today's release.
- Finally remember to push your changes.

Solution:

Connect to the storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Switch to Root user

```
sudo -i
```

Navigate to the bare repository hooks directory

```
cd /opt/official.git/hooks
```

Create the post-update hook from sample (or new file)

```
cp post-update.sample post-update
```

Replace content with this script

```
#!/bin/sh
# post-update hook: always create today's release tag for master

DATE=$(date +%F)

# Check if master branch changed
CHANGED_MASTER=0
for ref in "$@"
do
    if echo "$ref" | grep -q "refs/heads/master"; then
        CHANGED_MASTER=1
        break
```

```
    fi
done

# If master changed or stdin is empty, attempt to create tag
if [ $CHANGED_MASTER -eq 1 ] || [ -z "$@" ]; then
    echo "Creating release tag for $DATE..."
    if ! git rev-parse "release-$DATE" >/dev/null 2>&1; then
        git tag -a "release-$DATE" -m "Release for $DATE"
    else
        echo "Tag release-$DATE already exists"
    fi
fi
```

Make the hook executable

```
chmod +x post-update
```

In your working clone, merge feature into master

```
cd /usr/src/kodekloudrepos/official
git checkout master
git merge --no-ff feature -m "Merge feature into master"
```

Push master to the bare repository to trigger the hook

```
git push origin master
```

Fetch tags in your clone and verify

```
git fetch --tags
git tag
git show release-$(date +%F)
```

Docker

Day 35: Install Docker Packages and Start Docker Service

Task Description:

The Nautilus DevOps team aims to containerize various applications following a recent meeting with the application development team. They intend to conduct testing with the following steps:

- Install `docker-ce` and `docker compose` packages on `App Server 3`.
- Initiate the `docker` service.

Solution:

Connect to the App server 03

```
ssh banner@stapp03  // BigGr33n
```

Switch to the root user

```
sudo -i
```

Check the os Information

```
cat /etc/os-release
```

Visit the Docker Official Installation page
https://docs.docker.com/engine/install/centos/

Uninstall old versions

```
sudo dnf remove docker docker-client docker-client-latest docker-common
docker-latest docker-latest-logrotate docker-logrotate docker-engine
```

Installation method

Set up the repository

```
sudo dnf -y install dnf-plugins-core
sudo dnf config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
```

Install Docker Engine
To install the latest version, run:

```
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
```

```
docker-compose-plugin
```

Start Docker Engine.

```
sudo systemctl start docker
sudo systemctl enable –now docker
sudo systemctl status docker
```

Verify installation:

```
docker --version
docker compose --version
```

Day 36: Deploy Nginx Container on Application Server

Task Description:

The Nautilus DevOps team is conducting application deployment tests on selected application servers. They require a nginx container deployment on Application Server 2. Complete the task with the following instructions:

On Application Server 2 create a container named nginx_2 using the nginx image with the alpine tag. Ensure container is in a running state.

Solution:

Connect to app server 02

```
ssh steve@stapp02  //Am3ric@
```

Run docker command

```
docker run -d --name nginx_2 nginx:alpine
```

Day 37: Copy File to Docker Container

Task Description:

The Nautilus DevOps team possesses confidential data on `App Server 1` in the `Stratos Datacenter`. A container named `ubuntu_latest` is running on the same server.

Copy an encrypted file `/tmp/nautilus.txt.gpg` from the docker host to the `ubuntu_latest` container located at `/home/`. Ensure the file is not modified during this operation.

Solution:

Connect to app server 01

```
ssh tony@stapp01  //Ir0nM@n
```

Check the Docker container

```
docker ps
```

Then copy the file by docker cp command

```
docker cp /tmp/nautilus.txt.gpg ubuntu_latest:/home/
```

Verify inside the container

```
docker exec -it ubuntu_latest bash
```

Check the file:

```
ls -l /home/nautilus.txt.gpg
```

Day 38: Pull Docker Image

Task Description:

Nautilus project developers are planning to start testing on a new project. As per their meeting with the DevOps team, they want to test containerized environment application features. As per details shared with DevOps team, we need to accomplish the following task:

Pull `busybox:musl` image on `App Server 2` in Stratos DC and re-tag (create new tag) this image as `busybox:blog`

Solution:

Connect to app server 02

```
ssh steve@stapp02  //Am3ric@
```

Pull the Image

```
docker pull busybox:musl
```

Tag the docker image

```
docker tag busybox:musl busybox:blog
```

Day 39: Create a Docker Image From Container

Task Description:

One of the Nautilus developer was working to test new changes on a container. He wants to keep a backup of his changes to the container. A new request has been raised for the DevOps team to create a new image from this container. Below are more details about it:

Create an image `news:datacenter` on `Application Server 1` from a container `ubuntu_latest` that is running on same server.

Solution:

Connect to app server 01

```
ssh tony@stapp01  //Ir0nM@n
```

Check the ubuntu_latest container

```
docker ps
```

Commit the container into a new image:
Use docker commit to save the current state of the container into an image.

```
docker commit ubuntu_latest news:datacenter
```

- ubuntu_latest → container name (or ID).
- news:datacenter → new image name and tag.

Verify the new image:

```
 docker images
```

Day 40: Docker EXEC Operations

Task Description:

One of the Nautilus DevOps team members was working to configure services on a `kkloud` container that is running on `App Server 2` in `Stratos Datacenter`. Due to some personal work he is on PTO for the rest of the week, but we need to finish his pending work ASAP. Please complete the remaining work as per details given below:

a. Install `apache2` in `kkloud` container using `apt` that is running on `App Server 2` in `Stratos Datacenter`.
b. Configure Apache to listen on port `3001` instead of default `http` port. Do not bind it to listen on specific IP or hostname only, i.e it should listen on localhost, 127.0.0.1, container ip, etc.
c. Make sure Apache service is up and running inside the container. Keep the container in running state at the end.

Solution:

Connect to app server 01

```
ssh steve@stapp02  //Am3ric@
```

Check the kkloud container

```
docker ps
```

If the container is not running, start it first:

```
sudo docker start kkloud
```

Access the running kkloud container

```
sudo docker exec -it kkloud /bin/bash
```

Update package index and install apache2
Inside the container:

```
apt update && apt install -y apache2
```

Verify installation:

```
apache2 -v
```

Change Apache Port to 3001
By default Apache listens on port 80, we need to make it 3001.
Edit the ports config file:

```
vi /etc/apache2/ports.conf
```

Change: **Listen 80** to: **Listen 3001**

Also edit the default site configuration:

```
vi /etc/apache2/sites-available/000-default.conf
```

Change: **<VirtualHost *:80>** to: **<VirtualHost *:3001>**

Restart Apache
Enable and restart inside container:

```
service apache2 restart
```

Check status:

```
service apache2 status
```

Verify Apache is running on port 3001
Run:

```
ss -tlnp | grep 3001
```

or

```
netstat -tlnp | grep apache2
```

You should see Apache listening on 0.0.0.0:3001.

Day 41: Write a Docker file

Task Description:

As per recent requirements shared by the Nautilus application development team, they need custom images created for one of their projects. Several of the initial testing requirements are already been shared with DevOps team. Therefore, create a docker file `/opt/docker/Dockerfile` (please keep `D` capital of Dockerfile) on `App server 3` in `Stratos DC` and configure to build an image with the following requirements:

a. Use `ubuntu:24.04` as the base image.
b. Install `apache2` and configure it to work on `5003` port. (do not update any other Apache configuration settings like document root etc).

Solution:

Connect to app server 03

```
ssh banner@stapp03  //BigGr33n
```

Navigate to the directory

```
cd /opt/docker
```

Create Dockerfile

```
touch Dockerfile
```

Edit the file

```
nano Dockerfile
```

Add the following content

```
FROM ubuntu:24.04

RUN apt-get update && apt-get install -y apache2

RUN sed -i 's/Listen 80/Listen 5003/g' /etc/apache2/ports.conf

RUN sed -i 's/<VirtualHost \*:80>/<VirtualHost \*:5003>/g'
/etc/apache2/sites-available/000-default.conf

EXPOSE 5003

CMD ["apachectl", "-D", "FOREGROUND"]
```

Day 42: Create an Docker Network

Task Description:

The Nautilus DevOps team needs to set up several docker environments for different applications. One of the team members has been assigned a ticket where he has been asked to create some docker networks to be used later. Complete the task based on the following ticket description:

a. Create a docker network named as `official` on App Server `3` in `Stratos DC`.
b. Configure it to use `bridge` drivers.
c. Set it to use subnet `172.28.0.0/24` and iprange `172.28.0.0/24`.

Solution:

Connect to app server 03

```
ssh banner@stapp03  //BigGr33n
```

Run the Docker Network Command

```
sudo docker network create official --driver bridge --subnet 172.28.0.0/24
--ip-range 172.28.0.0/24
```

Check created network by,

```
docker network list
```

Day 43: Docker Ports Mapping

Task Description:

The Nautilus DevOps team is planning to host an application on a nginx-based container. There are number of tickets already been created for similar tasks. One of the tickets has been assigned to set up a nginx container on `Application Server 2` in `Stratos Datacenter`. Please perform the task as per details mentioned below:

a. Pull `nginx:stable` docker image on `Application Server 2`.
b. Create a container named `media` using the image you pulled.
c. Map host port `5000` to container port `80`. Please keep the container in running state.

Solution:

Connect to app server 02

```
ssh steve@stapp02  //Am3ric@
```

Pull the ***nginx:stable*** image

```
docker pull nginx:stable
```

Run the Container

```
docker run -d -p 5000:80 --name media nginx:stable
```

Check the container

```
docker ps
```

Day 44: Write a Docker Compose File

Task Description:

The Nautilus application development team shared static website content that needs to be hosted on the `httpd` web server using a containerised platform. The team has shared details with the DevOps team, and we need to set up an environment according to those guidelines. Below are the details:

a. On `App Server 3` in `Stratos DC` create a container named `httpd` using a docker compose file `/opt/docker/docker-compose.yml` (please use the exact name for file).
b. Use `httpd` (preferably `latest` tag) image for container and make sure container is named as `httpd`; you can use any name for service.
c. Map `80` number port of container with port `8084` of docker host.
d. Map container's `/usr/local/apache2/htdocs` volume with `/opt/data` volume of docker host which is already there. (please do not modify any data within these locations).

Solution:

Connect to app server 03

```
ssh banner@stapp03  //BigGr33n
```

Go to the directory

```
cd /opt/docker/
```

create Docker Compose file

```
touch docker-compose.yml
```

Edit the compose file

```
sudo vi docker-compose.yml
```

Add the Content on the file

```
services:
  apache:
    image: httpd:latest
    container_name: httpd
    ports:
    - '8084:80'
    volumes:
    - /opt/data:/usr/local/apache2/htdocs
```

Day 45: Resolve Dockerfile Issues

Task Description:

The Nautilus DevOps team is working to create new images per requirements shared by the development team. One of the team members is working to create a `Dockerfile` on `App Server 1` in `Stratos DC`. While working on it she ran into issues in which the docker build is failing and displaying errors. Look into the issue and fix it to build an image as per details mentioned below:

a. The `Dockerfile` is placed on `App Server 1` under `/opt/docker` directory.
b. Fix the issues with this file and make sure it is able to build the image.
c. Do not change base image, any other valid configuration within Dockerfile, or any of the data been used — for example, index.html.
`Note:` Please note that once you click on `FINISH` button all the existing containers will be destroyed and new image will be built from your `Dockerfile`.

Solution:

Connect to app server 01

```
ssh tony@stapp01  //Ir0nM@n
```

Go to the directory

```
cd /opt/docker/
```

In Dockerfile there is a path issue of the certificate and static html file and Change the COPY command.

Fixed Dockerfile

```
FROM httpd:2.4.43
RUN sed -i "s/Listen 80/Listen 8080/g" /usr/local/apache2/conf/httpd.conf
RUN sed -i '/LoadModule\ ssl_module modules\/mod_ssl.so/s/^#//g'
/usr/local/apache2/conf/httpd.conf
RUN sed -i '/LoadModule\ socache_shmcb_module
modules\/mod_socache_shmcb.so/s/^#//g' /usr/local/apache2/conf/httpd.conf
RUN sed -i '/Include\ conf\/extra\/httpd-ssl.conf/s/^#//g'
/usr/local/apache2/conf/httpd.conf
COPY certs/server.crt /usr/local/apache2/conf/server.crt
COPY certs/server.key /usr/local/apache2/conf/server.key
COPY html/index.html /usr/local/apache2/htdocs/
```

How to Test
On App Server 1:

```
cd /opt/docker
docker build -t custom-httpd .
```

If the Image successfully builds then Dockerfile is Corrected.

Day 46: Deploy an App on Docker Containers

Task Description:

The Nautilus Application development team recently finished development of one of the apps that they want to deploy on a containerized platform. The Nautilus Application development and DevOps teams met to discuss some of the basic pre-requisites and requirements to complete the deployment. The team wants to test the deployment on one of the app servers before going live and set up a complete containerized stack using a docker compose file. Below are the details of the task:

- On `App Server 1` in `Stratos Datacenter` create a docker compose file `/opt/finance/docker-compose.yml` (should be named exactly).
- The compose should deploy two services (web and DB), and each service should deploy a container as per details below:

For web service:

a. Container name must be php_blog.
b. Use image php with any apache tag. Check here for more details.
c. Map php_blog container's port 80 with host port 8087
d. Map php_blog container's /var/www/html volume with host volume /var/www/html.

For DB service:

a. Container name must be `mysql_blog`.
b. Use image `mariadb` with any tag (preferably `latest`). Check here for more details.
c. Map `mysql_blog` container's port `3306` with host port `3306`
d. Map `mysql_blog` container's `/var/lib/mysql` volume with host volume `/var/lib/mysql`.
e. Set MYSQL_DATABASE=`database_blog` and use any custom user (except root) with some complex password for DB connections.

- After running docker-compose up you can access the app with curl command `curl <server-ip or hostname>:8087/`

Solution:

Connect to app server 01

```
ssh tony@stapp01  //Ir0nM@n
```

Go to the directory

```
cd /opt/finance
```

Create Docker Compose file

```
touch docker-compose.yml
```

Edit the docker compose file

```
sudo vi docker-compose.yml
```

Add the Content on the File

```
services:
  web:
    container_name: php_blog
    image: php:apache
    ports:
      - "8087:80"
    volumes:
      - /var/www/html:/var/www/html
    depends_on:
      - db

  db:
    container_name: mysql_blog
    image: mariadb:latest
    ports:
      - "3306:3306"
    volumes:
      - /var/lib/mysql:/var/lib/mysql
    environment:
      MYSQL_DATABASE: database_blog
      MYSQL_USER: blog_user
      MYSQL_PASSWORD: ComplexP@ss123
      MYSQL_ROOT_PASSWORD: RootP@ss456
```

Day 47: Docker Python App

Task Description:

A python app needed to be Dockerized, and then it needs to be deployed on `App Server 2`. We have already copied a `requirements.txt` file (having the app dependencies) under `/python_app/src/` directory on `App Server 2`. Further complete this task as per details mentioned below:

- Create a `Dockerfile` under `/python_app` directory:
    - Use any `python` image as the base image.
    - Install the dependencies using `requirements.txt` file.
    - Expose the port `5001`.
    - Run the `server.py` script using `CMD`.

- Build an image named `nautilus/python-app` using this Dockerfile.
- Once image is built, create a container named `pythonapp_nautilus`:
    - Map port `5001` of the container to the host port `8094`.
- Once deployed, you can test the app using `curl` command on `App Server 2`.

`curl http://localhost:8094/`

Solution:

Connect to app server 02

```
ssh steve@stapp02  //Am3ric@
```

Go to the directory and check the requirement.txt file

```
cd /python_app/src
```

Go to the python app directory.

```
cd /python_app
```

Create Dockerfile

```
touch Dockerfile
```

Edit the Dockerfile

```
sudo nano Dockerfile
```

Add the Content on the File

```
FROM python:3.9-slim

WORKDIR /app
```

```
COPY src/requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY src/ .

EXPOSE 5001

CMD ["python", "server.py"]
```

Build the Image

```
docker build -t nautilus/python-app .
```

Run the Docker Container

```
docker run -d -p 8094:5001 --name pythonapp_nautilus nautilus/python-app
```

Curl to check the container

```
curl http://localhost:8094
```

Kubernetes

Day 48: Deploy Pods on Kubernetes Cluster

Task Description:

The Nautilus DevOps team is diving into Kubernetes for application management. One team member has a task to create a pod according to the details below:

- Create a pod named `pod-nginx` using the `nginx` image with the `latest` tag. Ensure to specify the tag as `nginx:latest`.
- Set the `app` label to `nginx_app`, and name the container as `nginx-container`.

`Note`: The `kubectl` utility on `jump_host` is configured to operate with the Kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Declarative way (YAML manifest)

Create a file named **pod-nginx.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
  labels:
    app: nginx_app
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
```

Apply it with:

```
kubectl apply -f pod-nginx.yaml
```

Day 49: Deploy Applications with Kubernetes Deployments

Task Description:

The Nautilus DevOps team is delving into Kubernetes for app management. One team member needs to create a deployment following these details:

Create a deployment named `nginx` to deploy the application `nginx` using the image `nginx:latest` (ensure to specify the tag)

`Note:` The `kubectl` utility on `jump_host` is set up to interact with the Kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Declarative way (YAML manifest)
Create a file named **deploy-nginx.yaml**:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Apply it with:

```
kubectl apply -f deploy-nginx.yaml
```

Day 50: Set Resource Limits in Kubernetes Pods

Task Description:

The Nautilus DevOps team has noticed performance issues in some Kubernetes-hosted applications due to resource constraints. To address this, they plan to set limits on resource utilization. Here are the details:

Create a pod named `httpd-pod` with a container named `httpd-container`. Use the `httpd` image with the `latest` tag (specify as `httpd:latest`). Set the following resource limits:

Requests: Memory: `15Mi`, CPU: `100m`

Limits: Memory: `20Mi`, CPU: `100m`

`Note:` The `kubectl` utility on `jump_host` is configured to operate with the Kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create a ***pod.yaml*** file and add the following content on it .

```
apiVersion: v1
kind: Pod
metadata:
  name: httpd-pod
spec:
  containers:
  - name: httpd-container
    image: httpd:latest
    resources:
      requests:
        memory: "15Mi"
        cpu: "100m"
      limits:
        memory: "20Mi"
        cpu: "100m"
    ports:
    - containerPort: 80
```

Then run

```
kubectl create -f pod.yaml
```

Day 51: Execute Rolling Updates in Kubernetes

Task Description:

An application currently running on the Kubernetes cluster employs the nginx web server. The Nautilus application development team has introduced some recent changes that need deployment. They've crafted an image `nginx:1.17` with the latest updates.

Execute a rolling update for this application, integrating the `nginx:1.17` image. The deployment is named `nginx-deployment`.

Ensure all pods are operational post-update.

`Note:` The `kubectl` utility on `jump_host` is set up to operate with the Kubernetes cluster

Solution:

This task is done on the jump host not in any app servers

First Check deployment yaml file

```
kubectl get deploy nginx-deployment -o yaml
```

This shows the description of deployment, notice the current image and container name.

After Viewing all the details, update the image in the deployment.

```
Kubectl set image deploy nginx-deployment nginx-container=nginx:1.17
```

Run the Watch command to see the live Rolling updates

```
watch kubectl get pods
```

Finally check status

```
Kubectl rollout status deploy nginx-deployment
```

Then check the deployment

```
Kubectl get deploy nginx-deployment -o yaml
```

Day 52: Revert Deployment to Previous Version in K8s

Task Description:

Earlier today, the Nautilus DevOps team deployed a new release for an application. However, a customer has reported a bug related to this recent release. Consequently, the team aims to revert to the previous version.

There exists a deployment named `nginx-deployment`; initiate a rollback to the previous revision.

`Note:` The `kubectl` utility on `jump_host` is configured to interact with the Kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

First check deployment history

```
kubectl rollout history deploy nginx-deployment
```

Then Roll Back to old version

```
Kubectl rollout undo deploy nginx-deployment
```

Then check Staus

```
kubectl rollout status deploy nginx-deployment
```

Day 53: Resolve VolumeMounts Issue in K8s

Task Description:

We encountered an issue with our Nginx and PHP-FPM setup on the Kubernetes cluster this morning, which halted its functionality. Investigate and rectify the issue:

The pod name is `nginx-phpfpm` and configmap name is `nginx-config`. Identify and fix the problem. Once resolved, copy `/home/thor/index.php` file from the `jump host` to the `nginx-container` within the nginx document root. After this, you should be able to access the website using `Website` button on the top bar.
`Note:` The `kubectl` utility on `jump_host` is configured to operate with the Kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Checked pod details

```
kubectl describe pod nginx-phpfpm
```

Inspected ConfigMap:

```
kubectl get configmap nginx-config -o yaml
```

On inspection:

- Nginx served files from ***/var/www/html***
- PHP-FPM container mounted the shared volume at ***/usr/share/nginx/html***

Because the two containers used different document root paths, PHP-FPM could not access files served by Nginx.

Export existing pod spec

```
kubectl get pod nginx-phpfpm -o yaml > /tmp/nginx-phpfpm.yaml
```

Edit the spec

```
vi /tmp/nginx-phpfpm.yaml
```

Find the php-fpm-container volumeMount:

```
  - mountPath: /usr/share/nginx/html
```

Replace with:

```
 - mountPath: /var/www/html
```

Delete old pod

```
kubectl delete pod nginx-phpfpm
```

Recreate pod

```
kubectl apply -f /tmp/nginx-phpfpm.yaml
```

Once pod is running again, copy the PHP file into Nginx’s document root:

```
kubectl cp /home/thor/index.php nginx-phpfpm:/var/www/html/index.php -c
nginx-container
```

Verification
Check pod status:

```
 kubectl get pods
```

nginx-phpfpm should be Running.
If using the lab's Website button → click to open.

Day 54: Kubernetes Shared Volumes

Task Description:

We are working on an application that will be deployed on multiple containers within a pod on Kubernetes cluster. There is a requirement to share a volume among the containers to save some temporary data. The Nautilus DevOps team is developing a similar template to replicate the scenario. Below you can find more details about it.

- Create a pod named `volume-share-nautilus`.
- For the first container, use image `debian` with `latest` tag only and remember to mention the tag i.e `debian:latest`, container should be named as `volume-container-nautilus-1`, and run a `sleep` command for it so that it remains in running state. Volume `volume-share` should be mounted at path `/tmp/official`.
- For the second container, use image `debian` with the `latest` tag only and remember to mention the tag i.e `debian:latest`, container should be named as `volume-container-nautilus-2`, and again run a `sleep` command for it so that it remains in running state. Volume `volume-share` should be mounted at path `/tmp/apps`.
- Volume name should be `volume-share` of type `emptyDir`.
- After creating the pod, exec into the first container i.e `volume-container-nautilus-1`, and just for testing create a file `official.txt` with any content under the mounted path of first container i.e `/tmp/official`.
- The file `official.txt` should be present under the mounted path `/tmp/apps` on the second container `volume-container-nautilus-2` as well, since they are using a shared volume.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create ***pod.yaml*** file

```
touch pod.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-share-nautilus
spec:
  containers:
```

```
    - name: volume-container-nautilus-1
      image: debian:latest
      command: ["bash", "-c", "sleep 3600"]
      volumeMounts:
        - name: volume-share
          mountPath: /tmp/official

    - name: volume-container-nautilus-2
      image: debian:latest
      command: ["bash", "-c", "sleep 3600"]
      volumeMounts:
        - name: volume-share
          mountPath: /tmp/apps

  volumes:
    - name: volume-share
      emptyDir: {}
```

Apply the manifest

```
kubectl apply -f pod.yaml
```

Verify the pod

```
kubectl get pods
```

Ensure ***volume-share-nautilus*** is in Running state.

Exec into the first container and create the file

```
kubectl exec -it volume-share-nautilus -c volume-container-nautilus-1 --
bash
```

Inside container:

```
echo "This is a shared file" > /tmp/official/official.txt
exit
```

Check from the second container

```
kubectl exec -it volume-share-nautilus -c volume-container-nautilus-2 --
bash
```

Inside container:

```
cat /tmp/apps/official.txt
```

You should see:

```
This is a shared file
```

Day 55: Kubernetes Sidecar Containers

Task Description:

We have a web server container running the nginx image. The access and error logs generated by the web server are not critical enough to be placed on a persistent volume. However, Nautilus developers need access to the last 24 hours of logs so that they can trace issues and bugs. Therefore, we need to ship the access and error logs for the web server to a log-aggregation service. Following the separation of concerns principle, we implement the Sidecar pattern by deploying a second container that ships the error and access logs from nginx. Nginx does one thing, and it does it well—serving web pages. The second container also specializes in its task—shipping logs. Since containers are running on the same Pod, we can use a shared emptyDir volume to read and write logs.

- Create a pod named `webserver`.
- Create an `emptyDir` volume `shared-logs`.
- Create two containers from `nginx` and `ubuntu` images with `latest` tag only and remember to mention tag i.e `nginx:latest`, nginx container name should be `nginx-container` and ubuntu container name should be `sidecar-container` on webserver pod.
- Add command on sidecar-container `"sh","-c","while true; do cat /var/log/nginx/access.log /var/log/nginx/error.log; sleep 30; done"`
- Mount the volume `shared-logs` on both containers at location `/var/log/nginx`, all containers should be up and running.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create ***pod.yaml*** file

```
touch pod.yaml
```

Add the Following content on the file.

```
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  containers:
    - name: nginx-container
```

```
      image: nginx:latest
      volumeMounts:
        - name: shared-logs
          mountPath: /var/log/nginx

    - name: sidecar-container
      image: ubuntu:latest
      command: ["sh", "-c", "while true; do cat /var/log/nginx/access.log
/var/log/nginx/error.log; sleep 30; done"]
      volumeMounts:
        - name: shared-logs
          mountPath: /var/log/nginx

  volumes:
    - name: shared-logs
      emptyDir: {}
```

Then run

```
kubectl apply -f pod.yaml
```

Verify pod and logs:

```
kubectl get pods
kubectl logs -f webserver -c sidecar-container
```

This ensures:

- ***nginx-container*** writes access/error logs to ***/var/log/nginx*** (shared volume).
- ***sidecar-container*** reads them every 30 seconds.

Day 56: Deploy Nginx Web Server on K8s Cluster

Task Description:

Some of the Nautilus team developers are developing a static website and they want to deploy it on Kubernetes cluster. They want it to be highly available and scalable. Therefore, based on the requirements, the DevOps team has decided to create a deployment for it with multiple replicas. Below you can find more details about it:

- Create a deployment using `nginx` image with `latest` tag only and remember to mention the tag i.e `nginx:latest`. Name it as `nginx-deployment`. The container should be named as `nginx-container`, also make sure replica counts are `3`.
- Create a `NodePort` type service named `nginx-service`. The nodePort should be `30011`.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create ***deploy.yaml*** file

```
touch deploy.yaml
```

Add the following content on the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx:latest
```

```
        ports:
          - containerPort: 80
```

Then run

```
kubectl apply -f deploy.yaml
```

Create Another file named ***nginx-service.yaml***

```
touch nginx-service.yaml
```

Add the following content on the file.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30011
```

Then run

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

Verify setup
Check the deployment and service status:

```
kubectl get deployments
kubectl get pods -o wide
kubectl get svc
```

We should see:

- A deployment named nginx-deployment with 3 replicas.
- A service named nginx-service of type NodePort exposing port 30011

Day 57: Print Environment Variables

Task Description:

The Nautilus DevOps team is working on setting up some pre-requisites for an application that will send the greetings to different users. There is a sample deployment that needs to be tested. Below is a scenario which needs to be configured on the Kubernetes cluster. Please find below more details about it.

- Create a `pod` named `print-envars-greeting`.
- Configure spec as, the container name should be `print-env-container` and use `bash` image.
- Create three environment variables:
  - `GREETING` and its value should be `Welcome to`
  - `COMPANY` and its value should be `DevOps`
  - `GROUP` and its value should be `Group`
- Use command `["/bin/sh", "-c", 'echo "$(GREETING) $(COMPANY) $(GROUP)"']` (please use this exact command), also set its `restartPolicy` policy to `Never` to avoid crash loop back.
- You can check the output using `kubectl logs -f print-envars-greeting` command.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create ***pod.yaml*** file

```
touch pod.yaml
```

Add the following content on the file.

```
apiVersion: v1
kind: Pod
metadata:
  name: print-envars-greeting
spec:
  containers:
    - name: print-env-container
      image: bash
      command: ["/bin/sh", "-c", "echo \"$(GREETING) $(COMPANY)
$(GROUP)\""]
      env:
        - name: GREETING
```

```
          value: "Welcome to"
        - name: COMPANY
          value: "DevOps"
        - name: GROUP
          value: "Group"
  restartPolicy: Never
```

Then run

```
kubectl apply -f pod.yaml
```

Check Pod status

```
kubectl get pods
```

View logs/output

```
kubectl logs -f print-envars-greeting
```

We should see the output:

```
Welcome to DevOps Group
```

Day 58: Deploy Grafana on K8s Cluster

Task Description:

The Nautilus DevOps team is planning to set up a Grafana tool to collect and analyze analytics from some applications. They are planning to deploy it on the Kubernetes cluster. Below you can find more details.

1) Create a deployment named `grafana-deployment-xfusion` using any grafana image for Grafana app. Set other parameters as per your choice.

2) Create `NodePort` type service with nodePort `32000` to expose the app.

`You need not to make any configuration changes inside the Grafana app once deployed, just make sure you are able to access the Grafana login page.`
`Note:` The `kubectl` on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create ***grafana-deployment.yaml*** file

```
touch grafana-deployment.yaml
```

Add the following content on the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana-deployment-xfusion
spec:
  replicas: 3
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
      - name: grafana
        image: grafana/grafana:latest
        ports:
```

```
        - containerPort: 3000
```

Then run

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

Create Another file named ***grafana-service.yaml***

```
touch grafana-service.yaml
```

Add the following content on the file.

```
apiVersion: v1
kind: Service
metadata:
  name: grafana-service
spec:
  type: NodePort
  selector:
    app: grafana
  ports:
    - port: 3000
      targetPort: 3000
      nodePort: 32000
```

Then run

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

Verify setup
Check the deployment and service status:

```
kubectl get deployments
kubectl get pods -o wide
kubectl get svc
```

We should see:

- A deployment named nginx-deployment with 3 replicas.
- A service named nginx-service of type NodePort exposing port 32000

Access Grafana
Now open your browser and go to:

```
http://<NodeIP>:32000
```

We should see the Grafana login page
(Default credentials are usually ***admin / admin***)

Day 59: Troubleshoot Deployment issues in K8s

Task Description:

Last week, the Nautilus DevOps team deployed a redis app on Kubernetes cluster, which was working fine so far. This morning one of the team members was making some changes in this existing setup, but he made some mistakes and the app went down. We need to fix this as soon as possible. Please take a look.

The deployment name is `redis-deployment`. The pods are not in running state right now, so please look into the issue and fix the same.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Check Pod Status

```
kubectl get pods
```

If they're in CrashLoopBackOff, Error, or ImagePullBackOff, we'll need to describe them next.

Describe the Pod

```
kubectl describe pod <pod-name>
```

Look for:

- Events at the bottom — image pull errors, failed probes, etc.
- Containers section — wrong image, bad env vars, etc.

Check Deployment Configuration

```
kubectl get deployment redis-deployment -o yaml
```

Check Logs

```
kubectl logs <pod-name>
```

This will show Redis startup errors (e.g., config file missing, permission issues, etc.)

Root Cause Analysis

After investigating the deployment and pod details, two configuration errors were identified:

1. Incorrect ConfigMap name, In the deployment YAML, the ConfigMap reference was set to:

```
name: redis-conig
```

However, the actual ConfigMap name should have been:

```
name: redis-config
```

This caused the following error:

```
MountVolume.SetUp failed for volume "config" : configmap "redis-conig" not
found
```

2. Incorrect Redis image name

The Redis image specified was:

```
image: redis:alpin
```

The correct Redis image tag should be:

```
image: redis:alpine
```

Because of the misspelled tag, Kubernetes was unable to pull a valid Redis image.

Steps Taken to Resolve

Edited Deployment

```
kubectl edit deployment redis-deployment
```

Updated:

```
image: redis:alpine
...
name: redis-config
```

Recreated ConfigMap if missing, otherwise leave configmap part

```
kubectl create configmap redis-config --from-literal=redis.conf="maxmemory
2mb"
```

Restarted the Deployment

```
kubectl rollout restart deployment redis-deployment
kubectl rollout status deployment redis-deployment
```

Verified the Pod Status

```
kubectl get pods
```

Day 60: Persistent Volumes in Kubernetes

Task Description:

The Nautilus DevOps team is working on a Kubernetes template to deploy a web application on the cluster. There are some requirements to create/use persistent volumes to store the application code, and the template needs to be designed accordingly. Please find more details below:

- Create a `PersistentVolume` named as `pv-datacenter`. Configure the `spec` as storage class should be `manual`, set capacity to `4Gi`, set access mode to `ReadWriteOnce`, volume type should be `hostPath` and set path to `/mnt/data` (this directory is already created, you might not be able to access it directly, so you need not to worry about it).
- Create a `PersistentVolumeClaim` named as `pvc-datacenter`. Configure the `spec` as storage class should be `manual`, request `3Gi` of the storage, set access mode to `ReadWriteOnce`.
- Create a `pod` named as `pod-datacenter`, mount the persistent volume you created with claim name `pvc-datacenter` at document root of the web server, the container within the pod should be named as `container-datacenter` using image `httpd` with `latest` tag only (remember to mention the tag i.e `httpd:latest`).
- Create a node port type service named `web-datacenter` using node port `30008` to expose the web server running within the pod.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

We need to create four Kubernetes resources: a PersistentVolume (PV), PersistentVolumeClaim (PVC), Pod, and Service.
Create the necessary files.

```
touch pv.yaml
touch pvc.yaml
touch pod.yaml
touch service.yaml
```

Open the ***pv.yaml*** file in editor

```
vi pv.yaml
```

Add the following content on the file

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: pv-datacenter
spec:
  storageClassName: manual
  capacity:
    storage: 4Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
```

Open the ***pvc.yaml*** file in editor

```
vi pvc.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-datacenter
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Open the ***pod.yaml*** file in editor

```
vi pod.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-datacenter
  labels:
    app: web-datacenter-app
spec:
  containers:
```

```
    - name: container-datacenter
      image: httpd:latest
      ports:
        - containerPort: 80
      volumeMounts:
        - name: html-volume
          mountPath: /usr/local/apache2/htdocs
  volumes:
    - name: html-volume
      persistentVolumeClaim:
        claimName: pvc-datacenter
```

Open the ***service.yaml*** file in editor

```
Vi service.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Service
metadata:
  name: web-datacenter
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30008
  selector:
    app: web-datacenter-app
```

Create the PV and PVC:

```
kubectl apply -f pv.yaml
kubectl apply -f pvc.yaml
```

Verify the PVC is Bound:

```
kubectl get pv,pvc
```

Both should show a STATUS of 'Bound'

Create the Pod and Service:

```
kubectl apply -f pod.yaml
```

```
kubectl apply -f web.yaml
```

Verify:

```
kubectl get pod pod-datacenter
kubectl get svc web-datacenter
```

Day 61: Init Containers in Kubernetes

Task Description:

There are some applications that need to be deployed on Kubernetes cluster and these apps have some pre-requisites where some configurations need to be changed before deploying the app container. Some of these changes cannot be made inside the images so the DevOps team has come up with a solution to use init containers to perform these tasks during deployment. Below is a sample scenario that the team is going to test first.

- Create a `Deployment` named as `ic-deploy-xfusion`.
- Configure `spec` as replicas should be `1`, labels `app` should be `ic-xfusion`, template's metadata lables `app` should be the same `ic-xfusion`.
- The `initContainers` should be named as `ic-msg-xfusion`, use image `ubuntu` with `latest` tag and use command `'/bin/bash'`, `'-c'` and `'echo Init Done - Welcome to xFusionCorp Industries > /ic/ecommerce'`. The volume mount should be named as `ic-volume-xfusion` and mount path should be `/ic`.
- Main container should be named as `ic-main-xfusion`, use image `ubuntu` with `latest` tag and use command `'/bin/bash'`, `'-c'` and `'while true; do cat /ic/ecommerce; sleep 5; done'`. The volume mount should be named as `ic-volume-xfusion` and mount path should be `/ic`.
- Volume to be named as `ic-volume-xfusion` and it should be an emptyDir type.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create the deploy.yaml file.

```
touch deploy.yaml
```

Open the ***deploy.yaml*** file in editor

```
vi deploy.yaml
```

Add the following content on the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ic-deploy-xfusion
  labels:
    app: ic-xfusion
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ic-xfusion
  template:
    metadata:
      labels:
        app: ic-xfusion
    spec:
      initContainers:
        - name: ic-msg-xfusion
          image: ubuntu:latest
          command:
            - '/bin/bash'
            - '-c'
            - 'echo Init Done - Welcome to xFusionCorp Industries >
/ic/ecommerce'
          volumeMounts:
            - name: ic-volume-xfusion
              mountPath: /ic

      containers:
        - name: ic-main-xfusion
          image: ubuntu:latest
          command:
            - '/bin/bash'
            - '-c'
            - 'while true; do cat /ic/ecommerce; sleep 5; done'
          volumeMounts:
            - name: ic-volume-xfusion
              mountPath: /ic

      volumes:
        - name: ic-volume-xfusion
          emptyDir: {}
```

Create the Deployment:

```
kubectl apply -f deploy.yaml
```

Verify the Deployment and Pod Status:

```
kubectl get deployment ic-deploy-xfusion
kubectl get pods -l app=ic-xfusion
```

The pod status should eventually show as `Running`.

Check the Logs (Verification):
You can check the logs of the main container to confirm that it's reading the message written by the init container.

```
POD_NAME=$(kubectl get pods -l app=ic-xfusion -o
jsonpath='{.items[0].metadata.name}')

kubectl logs $POD_NAME -c ic-main-xfusion
```

The output should repeatedly show:

`Init Done - Welcome to xFusionCorp Industries.`

Day 62: Manage Secrets in Kubernetes

Task Description:

The Nautilus DevOps team is working to deploy some tools in Kubernetes cluster. Some of the tools are licence based so that licence information needs to be stored securely within Kubernetes cluster. Therefore, the team wants to utilize Kubernetes secrets to store those secrets. Below you can find more details about the requirements:

- We already have a secret key file `news.txt` under `/opt` location on `jump host`. Create a `generic secret` named `news`, it should contain the password/license-number present in `news.txt` file.
- Also create a `pod` named `secret-nautilus`.
- Configure pod's `spec` as container name should be `secret-container-nautilus`, image should be `fedora` with `latest` tag (remember to mention the tag with image). Use `sleep` command for container so that it remains in running state. Consume the created secret and mount it under `/opt/demo` within the container.
- To verify you can exec into the container `secret-container-nautilus`, to check the secret key under the mounted path `/opt/demo`. Before hitting the `Check` button please make sure pod/pods are in running state, also validation can take some time to complete so keep patience.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create the Generic Secret

We will use the ***kubectl create secret generic*** command to create the Secret named ***news*** from the file ***/opt/news.txt*** on the jump host.

```
kubectl create secret generic news --from-file=/opt/news.txt
```

- The content of the file ***/opt/news.txt*** will be stored as a key-value pair within the Secret. By default, the key in the Secret will be the filename, which is ***news.txt***.

Create the pod.yaml file.

```
touch pod.yaml
```

Open the ***pod.yaml*** file in editor

```
vi pod.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-nautilus
spec:
  containers:
  - name: secret-container-nautilus
    image: fedora:latest
    command: ["/bin/bash", "-c", "sleep 3600"]
    volumeMounts:
    - name: news-secret-volume
      mountPath: /opt/demo
      readOnly: true
  volumes:
  - name: news-secret-volume
    secret:
      secretName: news
```

Apply the Pod Manifest

```
kubectl apply -f pod.yaml
```

Verification:

Check Pod Status:

```
kubectl get pods secret-nautilus
```

Exec into the container and list the files in the mounted path:

```
kubectl exec -it secret-nautilus -c secret-container-nautilus -- ls
/opt/demo
```

We should see the file ***news.txt*** listed

View the content of the secret file:

```
kubectl exec -it secret-nautilus -c secret-container-nautilus -- cat
/opt/demo/news.txt
```

This should display the content of the original ***/opt/news.txt*** file

Day 63: Deploy Iron Gallery App on Kubernetes

Task Description:

There is an iron gallery app that the Nautilus DevOps team was developing. They have recently customized the app and are going to deploy the same on the Kubernetes cluster. Below you can find more details:

- Create a namespace `iron-namespace-xfusion`
- Create a deployment `iron-gallery-deployment-xfusion` for `iron gallery` under the same namespace you created.
  - Labels `run` should be `iron-gallery`.
  - Replicas count should be `1`.
  - Selector's matchLabels `run` should be `iron-gallery`.
  - Template labels `run` should be `iron-gallery` under metadata.
  - The container should be named as `iron-gallery-container-xfusion`, use `kodekloud/irongallery:2.0` image (use exact image name / tag).
  - Resources limits for memory should be `100Mi` and for CPU should be `50m`.
  - First volumeMount name should be `config`, its mountPath should be `/usr/share/nginx/html/data`.
  - Second volumeMount name should be `images`, its mountPath should be `/usr/share/nginx/html/uploads`.
  - First volume name should be `config` and give it `emptyDir` and second volume name should be `images`, also give it `emptyDir`.
- Create a deployment `iron-db-deployment-xfusion` for `iron db` under the same namespace.
  - Labels `db` should be `mariadb`.
  - Replicas count should be `1`.
  - Selector's matchLabels `db` should be `mariadb`.
  - Template labels `db` should be `mariadb` under metadata.
  - The container name should be `iron-db-container-xfusion`, use `kodekloud/irondb:2.0` image (use exact image name / tag).
  - Define environment, set `MYSQL_DATABASE` its value should be `database_blog`, set `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD` value should be with some complex passwords for DB connections, and `MYSQL_USER` value should be any custom user (except root).
  - Volume mount name should be `db` and its mountPath should be `/var/lib/mysql`. Volume name should be `db` and give it an `emptyDir`.
- Create a service for `iron db` which should be named `iron-db-service-xfusion` under the same namespace. Configure spec as selector's db should be `mariadb`. Protocol should be `TCP`, port and targetPort should be `3306` and its type should be `ClusterIP`.
- Create a service for `iron gallery` which should be named `iron-gallery-service-xfusion` under the same namespace. Configure spec as

selector's run should be `iron-gallery`. Protocol should be `TCP`, port and targetPort should be `80`, nodePort should be `32678` and its type should be `NodePort`.

`Note:`

- We don't need to make connection b/w database and front-end now, if the installation page is coming up it should be enough for now.
- The `kubectl` on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

This task involves creating Kubernetes resources: Namespace, two Deployments, and two Services.

YAML manifest for the Kubernetes resources:

Create ***iron-app-resources.yaml*** file

```
touch iron-app-resources.yaml
```

Open the ***iron-app-resources.yaml*** file in editor

```
vi iron-app-resources.yaml
```

Add the following content on the file

```
---
# Create a namespace
apiVersion: v1
kind: Namespace
metadata:
  name: iron-namespace-xfusion

---
# Create iron-gallery deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iron-gallery-deployment-xfusion
  namespace: iron-namespace-xfusion
  labels:
    run: iron-gallery
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      run: iron-gallery
  template:
    metadata:
      labels:
        run: iron-gallery
    spec:
      containers:
      - name: iron-gallery-container-xfusion
        image: kodekloud/irongallery:2.0
        resources:
          limits:
            memory: "100Mi"
            cpu: "50m"
        volumeMounts:
        - name: config
          mountPath: /usr/share/nginx/html/data
        - name: images
          mountPath: /usr/share/nginx/html/uploads
      volumes:
      - name: config
        emptyDir: {}
      - name: images
        emptyDir: {}

---
# Create iron-db deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iron-db-deployment-xfusion
  namespace: iron-namespace-xfusion
  labels:
    db: mariadb
spec:
  replicas: 1
  selector:
    matchLabels:
      db: mariadb
  template:
    metadata:
```

```
      labels:
        db: mariadb
    spec:
      containers:
      - name: iron-db-container-xfusion
        image: kodekloud/irondb:2.0
        env:
        - name: MYSQL_DATABASE
          value: database_blog
        - name: MYSQL_ROOT_PASSWORD
          value: "S3cur3R00tP@ss"
        - name: MYSQL_PASSWORD
          value: "S3cur3Us3rP@ss"
        - name: MYSQL_USER
          value: iron_user_kk
        volumeMounts:
        - name: db
          mountPath: /var/lib/mysql
      volumes:
      - name: db
        emptyDir: {}

---
# Create iron-db service (ClusterIP)
apiVersion: v1
kind: Service
metadata:
  name: iron-db-service-xfusion
  namespace: iron-namespace-xfusion
spec:
  selector:
    db: mariadb
  ports:
  - protocol: TCP
    port: 3306
    targetPort: 3306
  type: ClusterIP

---
# Create iron-gallery service (NodePort)
apiVersion: v1
kind: Service
metadata:
```

```
  name: iron-gallery-service-xfusion
  namespace: iron-namespace-xfusion
spec:
  selector:
    run: iron-gallery
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 32678
  type: NodePort
```

Execution Command

```
kubectl apply -f iron-app-resources.yaml
```

Verification:

Check Namespace:

```
kubectl get ns iron-namespace-xfusion
```

Check Deployments:

```
kubectl get deploy -n iron-namespace-xfusion
```

Check Services:

```
kubectl get svc -n iron-namespace-xfusion
```

- The iron-gallery-service-xfusion service should show NodePort type and its port should include 32678.
- The iron-db-service-xfusion service should show ClusterIP type and its port should be 3306.

Day 64: Fix Python App Deployed on K8s Cluster

Task Description:

One of the DevOps engineers was trying to deploy a python app on Kubernetes cluster. Unfortunately, due to some mis-configuration, the application is not coming up. Please take a look into it and fix the issues. Application should be accessible on the specified nodePort.

- The deployment name is `python-deployment-devops`, its using `poroko/flask-demo-app`image. The deployment and service of this app is already deployed.
- nodePort should be `32345` and targetPort should be python flask app's default port.

`Note:` The `kubectl` on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

The issue was a combination of misconfiguration in both the Deployment and the Service, which prevented the Python Flask application from starting and being externally accessible.

The fix required two key steps: correcting the Pod container image and correcting the Service ports and selector.

Resources:

- Deployment: ***python-deployment-devops***
- Service: ***python-service-devops***
- Required Access: ***NodePort 32345***

The application was not accessible via the specified NodePort because the Pods failed to start, and the existing Service configuration did not correctly map external traffic to the running container.

Diagnosis and Root Cause Analysis:

Root Cause 1: ImagePullBackOff Error (Deployment Issue)

The Pods were stuck in an ImagePullBackOff state, indicating the container runtime could not pull the specified Docker image.

- Initial Manifest Error (Typo 1): The Deployment was configured with the image poroko/flask-app-demo.

- Correction Attempt Error (Typo 2): A subsequent fix attempt used poroko/flask-demo-appimage (as specified in the initial problem statement), which also resulted in ImagePullBackOff.
- Final Root Cause: The correct public image was determined to be poroko/flask-demo-app.

Root Cause 2: Service Misconfiguration

The Service, named python-service-devops, was not correctly configured to route traffic to the application Pods.

Resolution Steps:

The application was fixed by applying the following two commands/edits.

Step 1: Fix the Deployment Image
The container image in the ***python-deployment-devops*** Deployment was updated to the correct, non-typoed image name. This action triggered a new rolling update, allowing the Pod to pull the image and start the application on port 5000.

```
kubectl set image deployment/python-deployment-devops
python-container-devops=poroko/flask-demo-app
```

Step 2: Fix the Service Ports and Selector
The Service python-service-devops was edited to correctly map the external NodePort to the Pod's targetPort, while ensuring the selector matched the Pods' label.

```
kubectl edit service python-service-devops
```

The following changes were applied to the spec section:
spec.selector.app → python_app → Links the Service to the correct application Pods.

spec.ports[0].targetPort → 5000 → Directs traffic to the Flask app's listening port.

spec.ports[0].nodePort → 32345 →
Ensures external accessibility on the required port.

Verification
- Service Mapping: Traffic on 32345→Service 80→Pod 5000.
- Accessibility: Application is now accessible on ***http://<NODE_IP>:32345***.

Day 65: Deploy Redis Deployment on K8s

Task Description:

The Nautilus application development team observed some performance issues with one of the application that is deployed in Kubernetes cluster. After looking into number of factors, the team has suggested to use some in-memory caching utility for DB service. After number of discussions, they have decided to use Redis. Initially they would like to deploy Redis on kubernetes cluster for testing and later they will move it to production. Please find below more details about the task:
Create a redis deployment with following parameters:

- Create a `config map` called `my-redis-config` having `maxmemory 2mb` in `redis-config`.
- Name of the `deployment` should be `redis-deployment`, it should use `redis:alpine` image and container name should be `redis-container`. Also make sure it has only `1` replica.
- The container should request for `1` CPU.
- Mount `2` volumes:

a. An Empty directory volume called `data` at path `/redis-master-data`.

b. A configmap volume called `redis-config` at path `/redis-master`.

c. The container should expose the port `6379`.

- Finally, `redis-deployment` should be in an up and running state.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

Create ***configmap.yaml*** file

```
touch configmap.yaml
```

Open the ***configmap.yaml*** file in editor

```
vi configmap.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-redis-config
```

```
data:
  redis-config: |
    maxmemory 2mb
```

create the ConfigMap:

```
kubectl apply -f configmap.yaml
```

Create ***deploy.yaml*** file

```
touch deploy.yaml
```

Open the ***deploy.yaml*** file in editor

```
vi deploy.yaml
```

Add the following content on the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-deployment
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis-container
        image: redis:alpine
        resources:
          requests:
            cpu: "1"
        ports:
        - containerPort: 6379
        command: ["redis-server"]
        args: ["/redis-master/redis-config"]
        volumeMounts:
        - name: data
```

```
          mountPath: /redis-master-data
        - name: redis-config
          mountPath: /redis-master
      volumes:
      - name: data
        emptyDir: {}
      - name: redis-config
        configMap:
          name: my-redis-config
          items:
          - key: redis-config
            path: redis-config
```

create the Deployment,

```
kubectl apply -f deploy.yaml
```

Verification
Check the Deployment status:

```
kubectl get deployment redis-deployment
```

The Redis instance is now running in our Kubernetes cluster, using the configuration specified in the my-redis-config ConfigMap.

Day 66: Deploy MySQL on Kubernetes

Task Description:

A new MySQL server needs to be deployed on Kubernetes cluster. The Nautilus DevOps team was working on to gather the requirements. Recently they were able to finalize the requirements and shared them with the team members to start working on it. Below you can find the details:

Create a PersistentVolume `mysql-pv`, its capacity should be `250Mi`, set other parameters as per your preference.

Create a PersistentVolumeClaim to request this PersistentVolume storage. Name it as `mysql-pv-claim` and request a `250Mi` of storage. Set other parameters as per your preference.

Create a deployment named `mysql-deployment`, use any mysql image as per your preference. Mount the PersistentVolume at mount path `/var/lib/mysql`.

Create a `NodePort` type service named `mysql` and set nodePort to `30007`.

Create a secret named `mysql-root-pass` having a key pair value, where key is `password` and its value is `YUIidhb667`, create another secret named `mysql-user-pass` having some key pair values, where frist key is `username` and its value is `kodekloud_aim`, second key is `password` and value is `B4zNgHA7Ya`, create one more secret named `mysql-db-url`, key name is `database` and value is `kodekloud_db8`

Define some Environment variables within the container:
`name: MYSQL_ROOT_PASSWORD`, should pick value from secretKeyRef `name: mysql-root-pass` and `key: password`

`name: MYSQL_DATABASE`, should pick value from secretKeyRef `name: mysql-db-url` and `key: database`

`name: MYSQL_USER`, should pick value from secretKeyRef `name: mysql-user-pass` key `key: username`

`name: MYSQL_PASSWORD`, should pick value from secretKeyRef `name: mysql-user-pass` and `key: password`

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

This task involves creating several Kubernetes resources: Secrets, PersistentVolume (PV), PersistentVolumeClaim (PVC), Deployment, and Service.

Create Secrets
We'll create three secrets to store sensitive database information.

Create ***mysql-secrets.yaml*** file

```
touch mysql-secrets.yaml
```

Open the ***mysql-secrets.yaml*** file in editor

```
vi mysql-secrets.yaml
```

Add the following content on the file

```
---
# Create mysql-root-pass Secret
apiVersion: v1
kind: Secret
metadata:
  name: mysql-root-pass
type: Opaque
stringData:
  password: YUIidhb667

---
# Create mysql-user-pass secret
apiVersion: v1
kind: Secret
metadata:
  name: mysql-user-pass
type: Opaque
stringData:
  username: kodekloud_aim
  password: B4zNgHA7Ya

---
# Create mysql-db-url secret
apiVersion: v1
kind: Secret
metadata:
  name: mysql-db-url
type: Opaque
stringData:
  database: kodekloud_db8
```

Apply the secrets:

```
kubectl apply -f mysql-secrets.yaml
```

Create PersistentVolume (PV)
Create ***mysql-pv.yaml*** file

```
touch mysql-pv.yaml
```

Open the ***mysql-pv.yaml*** file in editor

```
vi mysql-pv.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 250Mi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/mysql"
  persistentVolumeReclaimPolicy: Retain
```

Apply the PV:

```
kubectl apply -f mysql-pv.yaml
```

Create PersistentVolumeClaim (PVC)
Create ***mysql-pvc.yaml*** file

```
touch mysql-pvc.yaml
```

Open the ***mysql-pvc.yaml*** file in editor

```
vi mysql-pvc.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

```
      storage: 250Mi
```

Apply the PVC:

```
kubectl apply -f mysql-pvc.yaml
```

Create Deployment
Create ***mysql-deploy.yaml*** file

```
touch mysql-deploy.yaml
```

Open the ***mysql-deploy.yaml*** file in editor

```
vi mysql-deploy.yaml
```

Add the following content on the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-root-pass
              key: password
```

```
        - name: MYSQL_DATABASE
          valueFrom:
            secretKeyRef:
              name: mysql-db-url
              key: database
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              name: mysql-user-pass
              key: username
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-user-pass
              key: password
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

Apply the Deployment:

```
kubectl apply -f mysql-deploy.yaml
```

Create Service

Create ***mysql-service.yaml*** file

```
touch mysql-service.yaml
```

Open the ***mysql-dservice.yaml*** file in editor

```
vi mysql-service.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  type: NodePort
  selector:
```

```
    app: mysql
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30007
```

Apply the Service:

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

Day 67: Deploy Guest Book App on Kubernetes

Task Description:

The Nautilus Application development team has finished development of one of the applications and it is ready for deployment. It is a guestbook application that will be used to manage entries for guests/visitors. As per discussion with the DevOps team, they have finalized the infrastructure that will be deployed on Kubernetes cluster. Below you can find more details about it.

`BACK-END TIER`

- Create a deployment named `redis-master` for Redis master.
  a.) Replicas count should be `1`.
  b.) Container name should be `master-redis-nautilus` and it should use image `redis`.
  c.) Request resources as `CPU` should be `100m` and `Memory` should be `100Mi`.
  d.) Container port should be redis default port i.e `6379`.
- Create a service named `redis-master` for Redis master. Port and targetPort should be Redis default port i.e `6379`.
- Create another deployment named `redis-slave` for Redis slave.
  a.) Replicas count should be `2`.
  b.) Container name should be `slave-redis-nautilus` and it should use `gcr.io/google_samples/gb-redisslave:v3` image.
  c.) Requests resources as `CPU` should be `100m` and `Memory` should be `100Mi`.
  d.) Define an environment variable named `GET_HOSTS_FROM` and its value should be `dns`.
  e.) Container port should be Redis default port i.e `6379`.
- Create another service named `redis-slave`. It should use Redis default port i.e `6379`.

`FRONT END TIER`

- Create a deployment named `frontend`.
  a.) Replicas count should be `3`.
  b.) Container name should be `php-redis-nautilus` and it should use `gcr.io/google-samples/gb-frontend@sha256:a908df8486ff66f2c4daa0d3d8a2fa09846a1fc8efd65649c0109695c7c5cbff` image.
  c.) Request resources as `CPU` should be `100m` and `Memory` should be `100Mi`.
  d.) Define an environment variable named as `GET_HOSTS_FROM` and its value should be `dns`.
  e.) Container port should be `80`.
- Create a service named `frontend`. Its `type` should be `NodePort`, port should be `80` and its `nodePort` should be `30009`.

Finally, you can check the `guestbook app` by clicking on `App` button.

`Note:` The `kubectl` utility on `jump_host` has been configured to work with the kubernetes cluster.

Solution:

This task is done on the jump host not in any app servers

We need to create a total of 2 Deployments and 2 Services in the Backend Tier, and in the Frontend Tier we have 1 Deployment and 1 Service to set up the guestbook application.

Back-End Tier

Redis Master Deployment
Create ***redis-master.yaml*** file

```
touch redis-master.yaml
```

Open the ***redis-master.yaml*** file in editor

```
vi redis-master.yaml
```

Add the following content on the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: guestbook
    tier: backend
    role: master
spec:
  replicas: 1
  selector:
    matchLabels:
      app: guestbook
      tier: backend
      role: master
  template:
    metadata:
      labels:
        app: guestbook
        tier: backend
        role: master
    spec:
      containers:
      - name: master-redis-nautilus
        image: redis
        resources:
```

```
          requests:
            cpu: "100m"
            memory: "100Mi"
        ports:
        - containerPort: 6379
```

Apply the Redis Master Deployment:

```
kubectl apply -f redis-master.yaml
```

Redis Master Service

Create ***redis-master-svc.yaml*** file

```
touch redis-master-svc.yaml
```

Open the ***redis-master-svc.yaml*** file in editor

```
vi redis-master-svc.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: guestbook
    tier: backend
    role: master
spec:
  selector:
    app: guestbook
    tier: backend
    role: master
  ports:
  - port: 6379
    targetPort: 6379
```

Apply the Redis Master Service:

```
kubectl apply -f redis-master-svc.yaml
```

Redis Slave Deployment

This deployment manages two Redis slave instances, which are configured to find the master via DNS.

Create ***redis-slave.yaml*** file

```
touch redis-slave.yaml
```

Open the ***redis-slave.yaml*** file in editor

```
vi redis-slave.yaml
```

Add the following content on the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-slave
  labels:
    app: guestbook
    tier: backend
    role: slave
spec:
  replicas: 2
  selector:
    matchLabels:
      app: guestbook
      tier: backend
      role: slave
  template:
    metadata:
      labels:
        app: guestbook
        tier: backend
        role: slave
    spec:
      containers:
      - name: slave-redis-nautilus
        image: gcr.io/google_samples/gb-redisslave:v3
        resources:
          requests:
            cpu: "100m"
            memory: "100Mi"
        ports:
        - containerPort: 6379
        env:
        - name: GET_HOSTS_FROM
          value: dns
```

Apply the Redis Slave Deployment:

```
kubectl apply -f redis-slave.yaml
```

Create ***redis-slave-svc.yaml*** file

```
touch redis-slave-svc.yaml
```

Open the ***redis-slave-svc.yaml*** file in editor

```
vi redis-slave-svc.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
  labels:
    app: guestbook
    tier: backend
    role: slave
spec:
  selector:
    app: guestbook
    tier: backend
    role: slave
  ports:
  - port: 6379
    targetPort: 6379
```

Apply the Redis Slave Service:

```
kubectl apply -f redis-slave-svc.yaml
```

Front-End Tier

Frontend Deployment:
This deployment manages three front-end instances, configured to find the Redis services via DNS.

Create ***frontend.yaml*** file

```
touch frontend.yaml
```

Open the ***frontend.yaml*** file in editor

```
vi frontend.yaml
```

Add the following content on the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis-nautilus
        image:
gcr.io/google-samples/gb-frontend@sha256:a908df8486ff66f2c4daa0d3d8a2fa0984
6a1fc8efd65649c0109695c7c5cbff
        resources:
          requests:
            cpu: "100m"
            memory: "100Mi"
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

Apply the Frontend Deployment:

```
kubectl apply -f frontend.yaml
```

Frontend Service:
This service exposes the front-end application externally using a NodePort.

Create ***frontend-svc.yaml*** file

```
touch frontend-svc.yaml
```

Open the ***frontend-svc.yaml*** file in editor

```
vi frontend-svc.yaml
```

Add the following content on the file

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: NodePort
  selector:
    app: guestbook
    tier: frontend
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30009
```

Apply the Frontend Service:

```
kubectl apply -f frontend-svc.yaml
```

Jenkins

Day 68: Setup Jenkins Server

Task Description:

The DevOps team at xFusionCorp Industries is initiating the setup of CI/CD pipelines and has decided to utilize Jenkins as their server. Execute the task according to the provided requirements:

1. Install `Jenkins` on the jenkins server using the `yum` utility only, and start its service.

    - If you face a timeout issue while starting the Jenkins service, refer to this.

2. Jenkin's admin user name should be `theadmin`, password should be `Adm!n321`, full name should be `James` and email should be `james@jenkins.stratos.xfusioncorp.com`.

`Note:`

1. To access the `jenkins` server, connect from the jump host using the `root` user with the password `S3curePass`.

2. After Jenkins server installation, click the `Jenkins` button on the top bar to access the Jenkins UI and follow on-screen instructions to create an admin user.

Solution:

Connect to the Jenkins server by,

```
ssh root@jenkins  //S3curePass
```

Install OpenJDK 17:
as root, run the command bellow to install java 17:

```
yum install java-17-openjdk -y
```

check the version :

```
java --version
```

Install Jenkins Service
Run the following command to download the repository file

```
wget -O /etc/yum.repos.d/jenkins.repo
http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo
```

This command downloads the repository file and place it on ***/etc/yum.repos.d/***.

Let's take a look on this file:

```
cat /etc/yum.repos.d/jenkins.repo
```

Expected Output

```
[jenkins]
name=Jenkins-stable
baseurl=http://pkg.jenkins.io/redhat-stable
gpgcheck=1
```

we need to import the key from jenkins:

```
rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

Now we can install Jenkins :

```
yum -y install jenkins
```

Start the Jenkins Service
Start the service and check its status.

```
systemctl daemon-reload
systemctl start jenkins
systemctl enable jenkins
systemctl status jenkins
```

Initial Setup

Access the Jenkins UI by navigating to the Jenkins server's URL (e.g., http://jenkins.stratos.xfusioncorp.com:8080).
The first screen will prompt for the Initial Admin Password.

On the Jenkins server, retrieve this password:

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

Copy the output and paste it into the administrator password field on the UI.

Create Admin User
After the plugins are installed, we will be prompted to create the first admin user. Use the following details as required:
Username: theadmin
Password: Adm!n321

Full Name: James
Email Address: james@jenkins.stratos.xfusioncorp.com

Complete the fields with the required values.
Click Save and Finish.
Proceed to the final confirmation screens to complete the Jenkins setup.

Day 69: Install Jenkins Plugins

Task Description:

The Nautilus DevOps team has recently setup a Jenkins server, which they want to use for some CI/CD jobs. Before that they want to install some plugins which will be used in most of the jobs. Please find below more details about the task

1. Click on the Jenkins button on the top bar to access the Jenkins UI. Login using username `admin` and `Adm!n321` password.
2. Once logged in, install the `Git` and `GitLab` plugins. Note that you may need to restart Jenkins service to complete the plugins installation, If required, opt to `Restart Jenkins when installation is complete and no jobs are running` on plugin installation/update page i.e `update centre`.

`Note:`
1. After restarting the Jenkins service, wait for the Jenkins login page to reappear before proceeding.
2. For tasks involving web UI changes, capture screenshots to share for review or consider using screen recording software like loom.com for documentation and sharing.

Solution:

Jenkins Login:
Click on the Jenkins button
On the Jenkins login page, Login with the provided credentials
Click the Sign in button

Plugin Installation:
Accessing the Plugin Manager
Once logged in, click on Manage Jenkins in the sidebar.
On the Manage Jenkins page, click on Manage Plugins.
Click on the Available tab.
Search Git and GitLab plugins
Recommended Option: Click Download now and install after restart.

Day 70: Configure Jenkins User Access

Task Description:

The Nautilus team is integrating Jenkins into their CI/CD pipelines. After setting up a new Jenkins server, they're now configuring user access for the development team, Follow these steps:

1. Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login with username `admin` and password `Adm!n321`.
2. Create a jenkins user named `jim` with the password`B4zNgHA7Ya`. Their full name should match `Jim`.
3. Utilize the `Project-based Matrix Authorization Strategy` to assign `overall read` permission to the `jim` user.
4. Remove all permissions for `Anonymous` users (if any) ensuring that the `admin` user retains overall `Administer` permissions.
5. For the existing job, grant `jim` user only `read` permissions, disregarding other permissions such as Agent, SCM etc.

`Note:`
1. You may need to install plugins and restart Jenkins service. After plugins installation, select `Restart Jenkins when installation is complete and no jobs are running` on plugin installation/update page.
2. After restarting the Jenkins service, wait for the Jenkins login page to reappear before proceeding. Avoid clicking `Finish` immediately after restarting the service.
3. Capture screenshots of your configuration for review purposes. Consider using screen recording software like `loom.com` for documentation and sharing.

Solution:

Access the Jenkins Portal and Log in using the provided credentials

Install Necessary Plugin
The **Project-based Matrix Authorization Strategy** usually requires the Authorization Strategy plugin (part of the core setup) and often works best with the Matrix Authorization Strategy plugin, which is typically installed by default. If the required strategy isn't available:

1. Go to Manage Jenkins → Manage Plugins.
2. Click the Available tab.
3. Search for **Matrix Authorization Strategy**.

Create the New User (jim)

Configure Global Security Strategy and Permissions

1. Go to Manage Jenkins → Configure Global Security.
2. In the **Authorization** section, select **Project-based Matrix Authorization Strategy**.
3. Configure the Global Permissions:
    - Add User: In the text box labeled "User/group to add," enter jim and click Add.
    - Assign Read Permission to jim: In the matrix, for the jim user row, check the Overall → Read permission box.
    - Check admin Permissions: Ensure the admin user retains the Overall → Administer permission (it should be checked by default if they were the initial user).
    - Remove Anonymous Permissions: Locate the row for Anonymous Users. Uncheck all permissions for this row, ensuring no boxes are checked.
4. Click Save at the bottom of the page.

Configure Job-Specific Permissions for **jim**

1. From the Jenkins dashboard, click on the **Helloworld** job.
2. Click Configure from the left-hand menu.
3. In the job configuration page, find the Authorization section (this is enabled by the **Project-based Matrix Authorization Strategy** you set globally).
4. Select the option Project-based security.
5. Add User: In the matrix, add the user jim.
6. Assign Read Permission to jim: In the jim user row, check the Read box
7. Click Save.

Day 71: Configure Jenkins Job for Package Installation

Task Description:

Some new requirements have come up to install and configure some packages on the Nautilus infrastructure under Stratos Datacenter. The Nautilus DevOps team installed and configured a new Jenkins server so they wanted to create a Jenkins job to automate this task. Find below more details and complete the task accordingly:

1. Access the Jenkins UI by clicking on the `Jenkins` button in the top bar. Log in using the credentials: username `admin` and password `Adm!n321`.
2. Create a new Jenkins job named `install-packages` and configure it with the following specifications:

- Add a string parameter named `PACKAGE`.
- Configure the job to install a package specified in the $PACKAGE parameter on the storage server within the Stratos Datacenter.

Solution:

Prerequisite for this task is to Install **SSH** Plugin.
Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password.

Under **Jenkins** > **Manage Jenkins** > **Manage Plugins** click **Available** and search for **SSH** plugin.
Select the **SSH** , **SSH Credentials** , **SSH Build Agent** plugin and click Install.
Restart Jenkins when installation is complete & refresh the login screen

Create SSH credentials in Jenkins:
Under **Jenkins** > **Manage Jenkins** > **Manage Credentials**, click **Global**
Under **Global** Stores scoped to Jenkins and **Add Credentials.**
Username: **natasha**
Password: **BI@kW**
ID: **storage**

Add SSH Hosts in Jenkins
Click **Jenkins** > **Manage Jenkins** > **Configure System**
Under SSH Remote Hosts click Add Host and provide the following values:
Hostname: **ststor01.stratos.xfusioncorp.com**
Port: **22**
Credentials: Choose **natasha** from the list
Click **Check Connection** to make sure connection is successful

Create the **Build** Job
Click New item and in the following screen:
Name: **install-packages** (Keep '**Freestyle Project**' as selected) and click **Ok**

Under Build Configuration Select **This project is parameterized** > **String Parameter**
Add a string parameter named **PACKAGE**
Default Value - **nano**

Under Build add a Build Step with **Execute shell script on remote host using SSH**

Under SSH Site select **natasha@ststor01.stratos.xfusioncorp.com:22**
In the command text area, provide the following.

```
echo 'Bl@kW' | sudo -S yum install -y $PACKAGE
```

Save the Configuration and Run the Job

Day 72: Jenkins Parameterized Builds

Task Description:

A new DevOps Engineer has joined the team and he will be assigned some Jenkins related tasks. Before that, the team wanted to test a simple parameterized job to understand basic functionality of parameterized builds. He is given a simple parameterized job to build in Jenkins. Please find more details below:

Click on the Jenkins button on the top bar to access the Jenkins UI. Login using username **admin** and password **Adm!n321**

1. Create a parameterized job which should be named as **parameterized-job**
2. Add a **string parameter** named **Stage**; its default value should be **Build**.
3. Add a **choice parameter** named **env**; its choices should be **Development**, **Staging** and **Production**.
4. Configure job to execute a shell command, which should echo both parameter values (you are passing in the job).
5. Build the Jenkins job at least once with **choice parameter** value **Staging** to make sure it passes.

Solution:

Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password.

Creating the Parameterized Job

1. From the Jenkins dashboard, click New Item (or Create a job).
2. In the Item name field, enter: **parameterized-job**
3. Select **Freestyle projec**t
4. Click OK.

Configuring Parameters

1. Under the General section, check the box for **This project is parameterized**.
2. Add the '**Stage**' String Parameter:
    - Click the **Add Parameter** dropdown and select **String Parameter**.
    - Name: **Stage**
    - Default Value: **Build**
    - Description: **The stage of the build process**
3. Add the '**env**' **Choice Paramete**r:
    - Click the **Add Parameter** dropdown and select **Choice Parameter**.
    - Name: **env**

In the Choices text area, enter each choice on a separate line:

```
Development
Staging
Production
```

- Description: **The target environment for the build**

Adding the Shell Command Build Step

1. Scroll down to the **Build** section.
2. Click Add build step and select **Execute shell**

In the Command text area, enter the shell script to echo the parameter values.

```
echo "The build stage is: ${Stage}"
echo "The target environment is: ${env}"
```

3. Click Apply, then Save.

Building the Job

1. Specify Parameter Values:
   - The Stage parameter should default to Build.
   - For the env parameter, select Staging from the dropdown menu (as required by the prompt).
2. Click the Build button

Day 73: Jenkins Scheduled Jobs

Task Description:

The devops team of xFusionCorp Industries is working on to set up a centralized logging management system to maintain and analyse server logs easily. Since it will take some time to implement, they wanted to gather some server logs on a regular basis. At least one of the app servers is having issues with the Apache server. The team needs Apache logs so that they can identify and troubleshoot the issues easily if they arise. So they decided to create a Jenkins job to collect logs from the server. Please create/configure a Jenkins job as per details mentioned below:
Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`
1. Create a Jenkins job named `copy-logs`.
2. Configure it to periodically build `every 6 minutes` to copy the Apache logs (`both access_log and error_logs`) from `App Server 1` (`from default logs location`) to location `/usr/src/security` on `Storage Server`.

`Note:`
Please make sure to define you cron expression like this `*/10 * * * *` (this is just an example to run a job every 10 minutes).

Solution:

Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password.

First we need to install SSH Plugin
Click **Jenkins** > **Manage Jenkins** > **Manage Plugins** and click the **Available tab**.
Search for **SSH** plugin and click Download now

Setup Credentials for SSH users
Under **Jenkins** > **Manage Jenkins** > **Manage Credentials**, click **Global** under Stores scoped to Jenkins and **Add Credentials**

Add SSH Hosts in Jenkins
Under **Jenkins** > **Manage Jenkins** > **Systems** > **SSH remote hosts**
Add two SSH Hosts **app server 01** and **Storage Server**

Create a Scheduled Build Job
Click New item and in the following screen:
Name: **copy-logs** (Keep '**Freestyle Project**' as selected) and click **Ok**

Under **Build Triggers**, select **Build Periodically** and provide the Schedule

In this example, the job is configured to run every 6 minutes

```
*/6 * * * *
```

Now, under **Build**, add a **Build Step** with **Execute shell script on remote host using SSH** and under SSH Site select **tony@stapp01:22**
Use below script

```
#!/bin/bash

APP_SERVER_PASSWORD="Ir0nM@n"
STORAGE_SERVER_PASSWORD="Bl@kW"

ACCESS_LOG="/var/log/httpd/access_log"
ERROR_LOG="/var/log/httpd/error_log"
DEST_DIR="/usr/src/security"

echo ${APP_SERVER_PASSWORD}| sudo -S yum install sshpass -y

echo ${APP_SERVER_PASSWORD} | sudo -S sshpass -p ${STORAGE_SERVER_PASSWORD}
scp -o StrictHostKeyChecking=no -r ${ACCESS_LOG}
natasha@ststor01:${DEST_DIR}

echo ${APP_SERVER_PASSWORD} | sudo -S sshpass -p ${STORAGE_SERVER_PASSWORD}
scp -o StrictHostKeyChecking=no -r ${ERROR_LOG}
natasha@ststor01:${DEST_DIR}
```

Validate Job success & login on storage server, check logs files

```
ssh natasha@ststor01 // Bl@kW
```

Go to the target directory

```
cd /usr/src/security
ls
```

Day 74: Jenkins Database Backup Job

Task Description:

There is a requirement to create a Jenkins job to automate the database backup. Below you can find more details to accomplish this task:

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`.

- Create a Jenkins job named `database-backup`.
- Configure it to take a database dump of the `kodekloud_db01` database present on the `Database server` in `Stratos Datacenter`, the database user is `kodekloud_roy` and password is `asdfgdsd`.
- The dump should be named in `db_$(date +%F).sql` format, where `date +%F` is the current date.
- Copy the `db_$(date +%F).sql` dump to the `Backup Server` under location `/home/clint/db_backups`.
- Further, schedule this job to run periodically at `*/10 * * * *` (please use this exact schedule format).

Solution:

Open Jenkins in new Tab
Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password.

Connect to the Jenkins server

```
ssh jenkins@jenkins  // j@rv!s
```

Generate Key Pair in Jenkins Server

```
ssh-keygen -t ed25519
```

Copying Keys to Database Server and Backup Server

```
ssh-copy-id peter@stdb01  // Sp!dy
ssh-copy-id clint@stbkp01  // H@wk3y3
```

This commands adds our public key to the server's **~/.ssh/authorized_keys**, enabling passwordless login.

Copy Private key in the Database Server and Backup Server

```
scp /var/lib/jenkins/.ssh/id_ed25519 peter@stdb01:/home/peter/.ssh/
scp /var/lib/jenkins/.ssh/id_ed25519 clint@stbkp01:/home/clint/.ssh/
```

Create a New Job

- Click “New Item”
- Enter job name: **database-backup**
- Choose “**Freestyle project**” → Click **OK**

Configure the Job
Scroll to **Build Triggers** → Check **Build periodically**.
In the Schedule field, paste:

```
*/10 * * * *
```

Under **Build**, scroll to Build Steps and choose:
Add **build step** → **Execute shell**

Then paste this shell script:

```
#!/bin/bash

DB_NAME="kodekloud_db01"
DB_USER="kodekloud_roy"
DB_PASS="asdfgdsd"

BACKUP_FILE="db_$(date +%F).sql"

ssh peter@stdb01 "mysqldump -u ${DB_USER} -p ${DB_PASS} ${DB_NAME} >
/tmp/${BACKUP_FILE}"

scp peter@stdb01:/tmp/${BACKUP_FILE} clint@stbkp01:/home/clint/db_backups/

ssh peter@stdb01 "rm -rf /tmp/${BACKUP_FILE}"
```

Save and Test

- Click Save.
- Click Build Now to test it.
- Check Console Output to verify:
    - Database dump is created.
    - File copied to **/home/clint/db_backups** on the Backup Server.

Day 75: Jenkins Slave Nodes

Task Description:

The Nautilus DevOps team has installed and configured a new Jenkins server in Stratos DC which they will use for CI/CD and for some automation tasks. There is a requirement to add all app servers as slave nodes in Jenkins so that they can perform tasks on these servers using Jenkins. Find below more details and accomplish the task accordingly.

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`.

1. Add all app servers as SSH build agent/slave nodes in Jenkins. Slave node name for `app server 1`, `app server 2` and `app server 3` must be `App_server_1`, `App_server_2`, `App_server_3` respectively.
2. Add labels as below:

```
App_server_1 : stapp01
App_server_2 : stapp02
App_server_3 : stapp03
```

3. Remote root directory for `App_server_1` must be `/home/tony/jenkins`, for `App_server_2` must be `/home/steve/jenkins` and for `App_server_3` must be `/home/banner/jenkins`.
4. Make sure slave nodes are online and working properly.

Solution:

Open Jenkins in new Tab
Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password.

We need to install **SSH**, **SSH Credentials** , **SSH Build Agent** Plugin
Click **Jenkins** > **Manage Jenkins** > **Manage Plugins** and click the **Available tab**.
Search for **SSH**, **SSH Credentials** , **SSH Build Agent** plugin and click Download now

We need to install Java on the target nodes (app servers).
SSH into App Server 1

```
ssh tony@stapp01
```

Install Java

```
sudo yum install -y java-17-openjdk
```

Verify installation

```
java -version
```

Add SSH Build Agent (Slave) Nodes

For App Server 1
In Jenkins UI, go to **Manage Jenkins → Nodes → New Node**
Enter node name: **App_server_1**
Select **Permanent Agent**, then click **OK**

Configure the node:
Description: **App Server 1 Node**
Number of executors: **1**
Remote root directory: **/home/tony/jenkins**
Labels: **stapp01**
Usage: "**Use this node as much as possible**"
Under Launch method, choose: **Launch agents via SSH**
Fill SSH details:
Host: **stapp01**
Credentials: Add credentials:
Click **Add → Jenkins → SSH Username with Password**
Username: **tony**
Password: Ir0nM@n
ID: stapp0
Save credentials.
Host Key Verification Strategy:Non verifying Verification Strategy

Save and click Launch agent (or Jenkins will auto-connect).

Verify Nodes Are Online
After saving each node configuration, Jenkins will attempt to connect via SSH
Wait until we see a green online indicator (✔) next to each node.
We can also verify by opening each node → click "**Log**" to ensure it says:
Agent successfully connected and online

Day 76: Jenkins Project Security

Task Description:

The xFusionCorp Industries has recruited some new developers. There are already some existing jobs on Jenkins and two of these new developers need permissions to access those jobs. The development team has already shared those requirements with the DevOps team, so as per details mentioned below grant required permissions to the developers.

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`.

- There is an existing Jenkins job named `Packages`, there are also two existing Jenkins users named `sam` with password `sam@pass12345` and `rohan` with password `rohan@pass12345`.
- Grant permissions to these users to access `Packages` job as per details mentioned below:
  a.) Make sure to select `Inherit permissions from parent ACL` under `inheritance strategy` for granting permissions to these users.
  b.) Grant mentioned permissions to `sam` user : `build`, `configure` and `read`.
  c.) Grant mentioned permissions to `rohan` user : `build`, `cancel`, `configure`, `read`, `update` and `tag`.

Solution:

Open Jenkins in new Tab
Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password.

Install Plugins
Go to: **Manage Jenkins** → **Manage Plugins** → **Available**
Install **Role-based Authorization Strategy** and **Matrix Authorization Strategy**

Enable Project-Based Permissions

1. Go to: **Manage Jenkins** → **Security**
2. Under Authorization, ensure:
   - **Project-based Matrix Authorization Strategy** is selected.
3. Click Save.

Configure Permissions for "**Packages**" Job
Navigate to the **Packages** job dashboard.
Click **Configure**
Scroll to **Enable project-based** security.

Assign Permissions
Add users:
Click Add user or group and type usernames exactly:

- sam
- rohan

Assign permission according to the ask in question

Day 77: Jenkins Deploy Pipeline

Task Description:

The development team of xFusionCorp Industries is working on to develop a new static website and they are planning to deploy the same on Nautilus App Servers using Jenkins pipeline. They have shared their requirements with the DevOps team and accordingly we need to create a Jenkins pipeline job. Please find below more details about the task:

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`.

Similarly, click on the `Gitea` button on the top bar to access the Gitea UI. Login using username `sarah` and password `Sarah_pass123`. There under user `sarah` you will find a repository named `web_app` that is already cloned on `Storage server` under `/var/www/html`. sarah is a developer who is working on this repository.

- Add a slave node named `Storage Server`. It should be labeled as `ststor01` and its remote root directory should be `/var/www/html`.
- We have already cloned repository on `Storage Server` under `/var/www/html`.
- Apache is already installed on all app Servers its running on port `8080`.
- Create a Jenkins pipeline job named `nautilus-webapp-job` (it must not be a `Multibranch pipeline`) and configure it to:
    - Deploy the code from `web_app` repository under `/var/www/html` on `Storage Server`, as this location is already mounted to the document root `/var/www/html` of app servers. The pipeline should have a single stage named `Deploy` (which is case sensitive) to accomplish the deployment.

LB server is already configured. You should be able to see the latest changes you made by clicking on the `App` button. Please make sure the required content is loading on the main URL `https://<LBR-URL>` i.e there should not be a sub-directory like `https://<LBR-URL>/web_app` etc.

Solution:

Open Jenkins in new Tab
Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password

Open Gitea in new Tab
Click on the Gitea button on the top bar. Login using the provided username and password.

We need to install **SSH**, **SSH Credentials** , **SSH Build Agent** Plugin
Click **Jenkins** > **Manage Jenkins** > **Manage Plugins** and click the **Available tab**.
Search for **SSH**, **SSH Credentials** , **SSH Build Agent** plugin and click Download now

We need to install Java on the target nodes (Storage server).
SSH into Storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Install Java

```
sudo yum install -y java-17-openjdk
```

Verify installation

```
java -version
```

Set the Permission of the target directory

```
sudo chmod 775 /var/www/html
sudo usermod -aG root natasha
```

Add SSH Build Agent (Slave) Node.

In Jenkins UI, go to **Manage Jenkins** → **Nodes** → **New Node**
Enter node name: **Storage Server**
Select **Permanent Agent**, then click **OK**

Configure the node:
Description: **Storage Server Node**
Number of executors: **1**
Remote root directory: **/var/www/html**
Labels: **ststor01**
Usage: "**Use this node as much as possible**"
Under Launch method, choose: **Launch agents via SSH**
Fill SSH details:
Host: **ststor01**
Credentials: Add credentials:
Click **Add** → **Jenkins** → **SSH Username with Password**
Username: **natasha**
Password: Bl@kW
ID: ststor01
Save credentials.
Host Key Verification Strategy: **Non verifying Verification Strategy**

Save and click Launch agent (or Jenkins will auto-connect)

We need to install **Pipeline**, **Pipeline Project** Plugin
Click **Jenkins** > **Manage Jenkins** > **Manage Plugins** and click the **Available tab**.
Search for **Pipeline**, **Pipeline Project** plugin and click Download now

Create Pipeline Job
Go to: **New Item → Pipeline**

- Name: **nautilus-webapp-job**
- Do NOT choose Multibranch Pipeline
- Restrict where this project can be run: Label Expression = **ststor01**

Add Credentials for the Gitea user
Under **Jenkins** > **Manage Jenkins** > **Manage Credentials**, click **Global** under Stores scoped to Jenkins and **Add Credentials**
Username: **sarah**
Password**: Sarah_pass123**
ID: **GIT_CRED**

Pipeline Script

```
pipeline {
    agent { label 'ststor01' }

    stages {
        stage('Deploy') {
            steps {
                sh 'cd /var/www/html && git pull'
            }
        }
    }
}
```

We need to install **Pipeline: GitHub**, **Git** and **SCM API** Plugin
Click **Jenkins** > **Manage Jenkins** > **Manage Plugins** and click the **Available tab**.
Search for **Pipeline: GitHub**, **Git** and **SCM API** plugin and click Download now

Day 78: Jenkins Conditional Pipeline

Task Description:

The development team of xFusionCorp Industries is working on to develop a new static website and they are planning to deploy the same on Nautilus App Servers using Jenkins pipeline. They have shared their requirements with the DevOps team and accordingly we need to create a Jenkins pipeline job. Please find below more details about the task:

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`.

Similarly, click on the `Gitea` button on the top bar to access the Gitea UI. Login using username `sarah` and password `Sarah_pass123`. There under user `sarah` you will find a repository named `web_app` that is already cloned on `Storage server` under `/var/www/html`. sarah is a developer who is working on this repository.

- Add a slave node named `Storage Server`. It should be labeled as `ststor01` and its remote root directory should be `/var/www/html`.
- We have already cloned repository on `Storage Server` under `/var/www/html`.
  Apache is already installed on all app Servers its running on port `8080`.
- Create a Jenkins pipeline job named `nautilus-webapp-job` (it must not be a `Multibranch pipeline`) and configure it to:

    - Add a string parameter named `BRANCH`.
    - It should conditionally deploy the code from `web_app` repository under `/var/www/html` on `Storage Server`, as this location is already mounted to the document root `/var/www/html` of app servers. The pipeline should have a single stage named `Deploy` (which is case sensitive) to accomplish the deployment.
    - The pipeline should be conditional, if the value `master` is passed to the `BRANCH` parameter then it must deploy the `master` branch, on the other hand if the value `feature` is passed to the `BRANCH` parameter then it must deploy the `feature` branch.

LB server is already configured. You should be able to see the latest changes you made by clicking on the `App` button. Please make sure the required content is loading on the main URL `https://<LBR-URL>` i.e there should not be a sub-directory like `https://<LBR-URL>/web_app` etc.

Solution:

Open Jenkins in new Tab
Click on the Jenkins button on the top bar. Login using the Jenkins admin user and password

Open Gitea in new Tab
Click on the Gitea button on the top bar. Login using the provided username and password.

We need to install Plugins
Click **Jenkins** > **Manage Jenkins** > **Manage Plugins** and click the **Available tab**.
Search for **SSH**, **SSH Credentials** , **SSH Build Agent, Pipeline**, **Pipeline Project, Pipeline: GitHub**, **Git** and **SCM API** plugin and click Download now.

We need to install Java on the target nodes (Storage server).
SSH into Storage Server

```
ssh natasha@ststor01  // Bl@kW
```

Install Java

```
sudo yum install -y java-17-openjdk
```

Verify installation

```
java -version
```

Set the Permission of the target directory

```
sudo chown -R natasha:natasha /var/www/html
```

Add SSH Build Agent (Slave) Node.

In Jenkins UI, go to **Manage Jenkins** → **Nodes** → **New Node**
Enter node name: **Storage Server**
Select **Permanent Agent**, then click **OK**

Configure the node:
Description: **Storage Server Node**
Number of executors: **1**
Remote root directory: **/var/www/html**
Labels: **ststor01**
Usage: "**Use this node as much as possible**"
Under Launch method, choose: **Launch agents via SSH**
Fill SSH details:
Host: **ststor01**
Credentials: Add credentials:
Click **Add** → **Jenkins** → **SSH Username with Password**
Username: **natasha**
Password: Bl@kW
ID: ststor01
Save credentials.
Host Key Verification Strategy: **Non verifying Verification Strategy**

Save and click Launch agent (or Jenkins will auto-connect)

Create Pipeline Job
Job Name: **nautilus-webapp-job**
Job Type: Pipeline (Not Multibranch)
Add a parameter:

1. Build **Parameters** → **Add Parameter** → **String Parameter**
2. Name: **BRANCH**
3. Default value: **master** (optional)

Add Credentials for the Gitea user
Under **Jenkins** > **Manage Jenkins** > **Manage Credentials**, click **Global** under Stores scoped to Jenkins and **Add Credentials**
Username: **sarah**
Password**: Sarah_pass123**
ID: **GIT_CRED**

Add Pipeline Script
In the Pipeline section, choose Pipeline Script and paste:

```
pipeline {
    agent { label 'ststor01' }

    parameters {
        string(name: 'BRANCH', defaultValue: 'master', description: 'Branch
to deploy')
    }

    stages {
        stage('Deploy') {
            when {
                anyOf {
                    environment name: 'BRANCH', value: 'master'
                    environment name: 'BRANCH', value: 'feature'
                }
            }
            steps {
                echo "Deploying branch: ${params.BRANCH}"
                dir('/var/www/html') {
                    sh "git fetch origin"
                    sh "git checkout ${params.BRANCH}"
                    sh "git pull origin ${params.BRANCH}"
                }
```

```
            }
        }
    }
}
```

Day 79: Jenkins Deployment Job

Task Description:

The Nautilus development team had a meeting with the DevOps team where they discussed automating the deployment of one of their apps using Jenkins (the one in `Stratos Datacenter`). They want to auto deploy the new changes in case any developer pushes to the repository. As per the requirements mentioned below configure the required Jenkins job.

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and `Adm!n321` password.

Similarly, you can access the `Gitea UI` using `Gitea` button, username and password for Git is `sarah` and `Sarah_pass123` respectively. Under user `sarah` you will find a repository named `web` that is already cloned on the `Storage server` under sarah's home. `sarah` is a developer who is working on this repository.

1. Install `httpd` (whatever version is available in the yum repo by default) and configure it to serve on port `8080` on All app servers. You can make it part of your Jenkins job or you can do this step manually on all app servers.

2. Create a Jenkins job named `nautilus-app-deployment` and configure it in a way so that if anyone pushes any new change to the origin repository in `master` branch, the job should auto build and deploy the latest code on the `Storage server` under `/var/www/html` directory. Since `/var/www/html` on `Storage server` is shared among all apps.
Before deployment, ensure that the ownership of the `/var/www/html` directory is set to user `sarah`, so that Jenkins can successfully deploy files to that directory.

3. SSH into `Storage Server` using `sarah` user credentials mentioned above. Under sarah user's home you will find a cloned Git repository named `web`. Under this repository there is an `index.html` file, update its content to `Welcome to the xFusionCorp Industries`, then push the changes to the `origin` into `master` branch. This push must trigger your Jenkins job and the latest changes must be deployed on the servers, also make sure it deploys the entire repository content not only `index.html` file.

Click on the `App` button on the top bar to access the app, you should be able to see the latest changes you deployed. Please make sure the required content is loading on the main URL `https://<LBR-URL>` i.e there should not be any sub-directory like `https://<LBR-URL>/web` etc.

Solution:

Complete Solution Provided in below link:
https://www.nbtechsupport.co.in/2021/07/deployment-using-jenkins.html

Day 80: Jenkins Chained Builds

Task Description:

The DevOps team was looking for a solution where they want to restart Apache service on all app servers if the deployment goes fine on these servers in Stratos Datacenter. After having a discussion, they came up with a solution to use Jenkins chained builds so that they can use a downstream job for services which should only be triggered by the deployment job. So as per the requirements mentioned below configure the required Jenkins jobs.

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and `Adm!n321` password.

Similarly you can access `Gitea UI` on port `8090` and username and password for Git is `sarah` and `Sarah_pass123` respectively. Under user `sarah` you will find a repository named `web`.

Apache is already installed and configured on all app server so no changes are needed there. The doc root `/var/www/html` on all these app servers is shared among the Storage server under `/var/www/html` directory.

1. Create a Jenkins job named `nautilus-app-deployment` and configure it to pull change from the `master` branch of `web` repository on `Storage server` under `/var/www/html` directory, which is already a local git repository tracking the origin `web` repository. Since `/var/www/html` on `Storage server` is a shared volume so changes should auto reflect on all apps.

2. Create another Jenkins job named `manage-services` and make it a `downstream job` for `nautilus-app-deployment` job. Things to take care about this job are:
a. This job should restart `httpd` service on all app servers.

b. Trigger this job only if the `upstream job` i.e `nautilus-app-deployment` is stable.

`LB` server is already configured. Click on the `App` button on the top bar to access the app. You should be able to see the latest changes you made. Please make sure the required content is loading on the main URL `https://<LBR-URL>` i.e there should not be a sub-directory like `https://<LBR-URL>/web` etc.

Solution:

Complete Solution Provided in below link:
https://www.nbtechsupport.co.in/2021/07/creating-chained-builds-in-jenkins.html

Day 81: Jenkins Multistage Pipeline

Task Description:

The development team of xFusionCorp Industries is working on to develop a new static website and they are planning to deploy the same on Nautilus App Servers using Jenkins pipeline. They have shared their requirements with the DevOps team and accordingly we need to create a Jenkins pipeline job. Please find below more details about the task:

Click on the `Jenkins` button on the top bar to access the Jenkins UI. Login using username `admin` and password `Adm!n321`.

Similarly, click on the `Gitea` button on the top bar to access the Gitea UI. Login using username `sarah` and password `Sarah_pass123`.

There is a repository named `sarah/web` in Gitea that is already cloned on `Storage server` under `/var/www/html` directory.

- Update the content of the file `index.html` under the same repository to `Welcome to xFusionCorp Industries` and push the changes to the origin into the `master` branch.
- Apache is already installed on all app Servers its running on port `8080`.
- Create a Jenkins pipeline job named `deploy-job` (it must not be a `Multibranch pipeline` job) and pipeline should have two stages `Deploy` and `Test` (names are case sensitive). Configure these stages as per details mentioned below.

  a. The `Deploy` stage should deploy the code from `web` repository under `/var/www/html` on the `Storage Server`, as this location is already mounted to the document root `/var/www/html` of all app servers.

  b. The `Test` stage should just test if the app is working fine and website is accessible. Its up to you how you design this stage to test it out, you can simply add a `curl` command as well to run a curl against the LBR URL (`http://stlb01:8091`) to see if the website is working or not. Make sure this stage fails in case the website/app is not working or if the `Deploy` stage fails.

Click on the `App` button on the top bar to see the latest changes you deployed. Please make sure the required content is loading on the main URL `http://stlb01:8091` i.e there should not be a sub-directory like `http://stlb01:8091/web` etc.

Solution:

Complete Solution Provided in below link:
https://www.nbtechsupport.co.in/2021/07/jenkins-multi-stage-pipeline.html

Ansible

Day 82: Create Ansible Inventory for App Server Testing

Task Description:

The Nautilus DevOps team is testing Ansible playbooks on various servers within their stack. They've placed some playbooks under `/home/thor/playbook/` directory on the `jump host` and now intend to test them on `app server 1` in `Stratos DC`. However, an inventory file needs creation for Ansible to connect to the respective app. Here are the requirements:

a. Create an ini type Ansible inventory file `/home/thor/playbook/inventory` on `jump host`.
b. Include `App Server 1` in this inventory along with necessary variables for proper functionality.
c. Ensure the inventory hostname corresponds to the `server name` as per the wiki, for example `stapp01` for `app server 1` in `Stratos DC`.

`Note:` Validation will execute the playbook using the command `ansible-playbook -i inventory playbook.yml`. Ensure the playbook functions properly without any extra arguments.

Solution:

Go to the required directory

```
cd /home/thor/playbook
```

Create the inventory file:

```
vi inventory
```

Add the content inside the file:

```
[app_servers]
stapp01 ansible_host=172.16.238.10 ansible_user=tony
ansible_ssh_pass=Ir0nM@n
```

Run from the playbook directory:

```
ansible -i inventory app_servers -m ping
```

Then run the playbook:

```
ansible-playbook -i inventory playbook.yml
```

Day 83: Troubleshoot and Create Ansible Playbook

Task Description:

An Ansible playbook needs completion on the `jump host`, where a team member left off. Below are the details:

- The inventory file `/home/thor/ansible/inventory` requires adjustments. The playbook must run on `App Server 2` in `Stratos DC`. Update the inventory accordingly.
- Create a playbook `/home/thor/ansible/playbook.yml`. Include a task to create an empty file `/tmp/file.txt` on `App Server 2`.

`Note:` Validation will run the playbook using the command `ansible-playbook -i inventory playbook.yml`. Ensure the playbook works without any additional arguments.

Solution:

Fix the inventory file

```
cd /home/thor/ansible
vi inventory
```

Update the file with the content below:

```
[app_servers]
stapp02 ansible_host=172.16.238.11 ansible_user=steve
ansible_ssh_pass=Am3ric@ ansible_ssh_common_args='-o
StrictHostKeyChecking=no'
```

Test connectivity:

```
ansible -i inventory app_servers -m ping
```

If the Connection Successfull we receive **PONG** message

Create the playbook file

```
cd /home/thor/ansible
vi playbook.yml
```

Add the Content:

```
---
- name: Create file on App Server 2
  hosts: app_servers
  become: yes
  tasks:
    - name: Create an empty file
      file:
        path: /tmp/file.txt
        state: touch
```

This uses the file module

Run Playbook

```
ansible-playbook -i inventory playbook.yml
```

Day 84: Copy Data to App Servers using Ansible

Task Description:

The Nautilus DevOps team needs to copy data from the `jump host` to all `application servers` in `Stratos DC` using Ansible. Execute the task with the following details:

a. Create an inventory file `/home/thor/ansible/inventory` on `jump_host` and add all application servers as managed nodes.
b. Create a playbook `/home/thor/ansible/playbook.yml` on the `jump host` to copy the `/usr/src/data/index.html` file to all application servers, placing it at `/opt/data`.

`Note:` Validation will run the playbook using the command `ansible-playbook -i inventory playbook.yml`. Ensure the playbook functions properly without any extra arguments.

Solution:

Create the inventory file

```
cd /home/thor/ansible
vi inventory
```

Add the application servers under a group named **app_servers**:

```
[app_servers]
stapp01 ansible_host=172.16.238.10 ansible_user=tony
ansible_ssh_pass=Ir0nM@n ansible_ssh_common_args='-o
StrictHostKeyChecking=no'

stapp02 ansible_host=172.16.238.11 ansible_user=steve
ansible_ssh_pass=Am3ric@ ansible_ssh_common_args='-o
StrictHostKeyChecking=no'

stapp03 ansible_host=172.16.238.12 ansible_user=banner
ansible_ssh_pass=BigGr33n ansible_ssh_common_args='-o
StrictHostKeyChecking=no'
```

Test connectivity:

```
ansible -i inventory app_servers -m ping
```

Create the playbook

```
cd /home/thor/ansible
vi playbook.yml
```

Add the Content:

```
---
- name: Copy index.html to application servers
  hosts: app_servers
  become: yes

  tasks:
    - name: Create destination directory
      file:
        path: /opt/data
        state: directory

    - name: Copy file to all application servers
      copy:
        src: /usr/src/data/index.html
        dest: /opt/data/index.html
```

Run Ansibe Playbook

```
cd /home/thor/ansible
ansible-playbook -i inventory playbook.yml
```

Day 85: Create Files on App Servers using Ansible

Task Description:

The Nautilus DevOps team is testing various Ansible modules on servers in `Stratos DC`. They're currently focusing on file creation on remote hosts using Ansible. Here are the details:

a. Create an inventory file `~/playbook/inventory` on `jump host` and include `all app servers`.
b. Create a playbook `~/playbook/playbook.yml` to create a blank file `/usr/src/nfsdata.txt` on `all app servers`.
c. Set the permissions of the `/usr/src/nfsdata.txt` file to `0644`.
d. Ensure the user/group owner of the `/usr/src/nfsdata.txt` file is `tony` on `app server 1`, `steve` on `app server 2` and `banner` on `app server 3`.

`Note:` Validation will execute the playbook using the command `ansible-playbook -i inventory playbook.yml`, so ensure the playbook functions correctly without any additional arguments.

Solution:

Create the inventory file

```
cd /home/thor/playbook
vi inventory
```

Add the application servers under a group named **app_servers**:

```
[app_servers]
stapp01 ansible_host=172.16.238.10 ansible_user=tony
ansible_ssh_pass=Ir0nM@n ansible_ssh_common_args='-o
StrictHostKeyChecking=no'

stapp02 ansible_host=172.16.238.11 ansible_user=steve
ansible_ssh_pass=Am3ric@ ansible_ssh_common_args='-o
StrictHostKeyChecking=no'

stapp03 ansible_host=172.16.238.12 ansible_user=banner
ansible_ssh_pass=BigGr33n ansible_ssh_common_args='-o
StrictHostKeyChecking=no'
```

Test connectivity:

```
ansible -i inventory app_servers -m ping
```

Create the playbook

```
cd /home/thor/playbook
vi /playbook.yml
```

Add the Content:

```
---
- name: Create file and set correct permissions and ownership
  hosts: app_servers
  become: yes

  tasks:
    - name: Create blank file with mode 0644
      file:
        path: /usr/src/nfsdata.txt
        state: touch
        mode: '0644'

    - name: Set owner to tony on stapp01
      file:
        path: /usr/src/nfsdata.txt
        owner: tony
        group: tony
      when: inventory_hostname == "stapp01"

    - name: Set owner to steve on stapp02
      file:
        path: /usr/src/nfsdata.txt
        owner: steve
        group: steve
      when: inventory_hostname == "stapp02"

    - name: Set owner to banner on stapp03
      file:
        path: /usr/src/nfsdata.txt
        owner: banner
        group: banner
      when: inventory_hostname == "stapp03"
```

Run Ansibe Playbook

```
cd /home/thor/playbook
ansible-playbook -i inventory playbook.yml
```

Day 86: Ansible Ping Module Usage

Task Description:

The Nautilus DevOps team is planning to test several Ansible playbooks on different app servers in `Stratos DC`. Before that, some pre-requisites must be met. Essentially, the team needs to set up a password-less SSH connection between Ansible controller and Ansible managed nodes. One of the tickets is assigned to you; please complete the task as per details mentioned below:
a. `Jump host` is our Ansible controller, and we are going to run Ansible playbooks through `thor` user from `jump host`.
b. There is an inventory file `/home/thor/ansible/inventory` on `jump host`. Using that inventory file test `Ansible ping` from `jump host` to `App Server 3`, make sure ping works.

Solution:

Go to ansible directory

```
cd /home/thor/ansible
```

See the Content of Inventory file,

```
cat inventory
```

The output of the file

```
stapp01 ansible_host=172.16.238.10 ansible_ssh_pass=Ir0nM@n
stapp02 ansible_host=172.16.238.11 ansible_ssh_pass=Am3ric@
stapp03 ansible_host=172.16.238.12 ansible_ssh_pass=BigGr33n
```

Test connectivity:

```
ansible -i inventory stapp03 -m ping
```

Error occur, Connection fail

Note that ansible user is missing in all alias so we need to add the user

```
stapp01 ansible_host=172.16.238.10 ansible_user=tony
ansible_ssh_pass=Ir0nM@n
stapp02 ansible_host=172.16.238.11 ansible_user=steve
ansible_ssh_pass=Am3ric@
stapp03 ansible_host=172.16.238.12 ansible_user=banner
ansible_ssh_pass=BigGr33n
```

Test connectivity:

```
ansible -i inventory stapp03 -m ping
```

Now the Connection established

Day 87: Ansible Install Package

Task Description:

The Nautilus Application development team wanted to test some applications on app servers in `Stratos Datacenter`. They shared some pre-requisites with the DevOps team, and packages need to be installed on app servers. Since we are already using Ansible for automating such tasks, please perform this task using Ansible as per details mentioned below:

- Create an inventory file `/home/thor/playbook/inventory` on `jump host` and add all app servers in it.
- Create an Ansible playbook `/home/thor/playbook/playbook.yml` to install `samba` package on `all app servers` using Ansible `yum` module.
- Make sure user `thor` should be able to run the playbook on `jump host`.

`Note:` Validation will try to run playbook using command `ansible-playbook -i inventory playbook.yml` so please make sure playbook works this way, without passing any extra arguments.

Solution:

Create the inventory file

```
cd /home/thor/playbook
vi inventory
```

Add the application servers under a group named **app_servers**:

```
[app_servers]
stapp01 ansible_host=172.16.238.10 ansible_user=tony
ansible_ssh_pass=Ir0nM@n

stapp02 ansible_host=172.16.238.11 ansible_user=steve
ansible_ssh_pass=Am3ric@

stapp03 ansible_host=172.16.238.12 ansible_user=banner
ansible_ssh_pass=BigGr33n
```

Test connectivity:

```
ansible -i inventory app_servers -m ping
```

Create the playbook

```
cd /home/thor/playbook
vi playbook.yml
```

Add the Content:

```
---
- name: Install samba on all app servers
  hosts: app_servers
  become: yes
  tasks:
    - name: Install samba package
      yum:
        name: samba
        state: present
```

Run Ansibe Playbook

```
cd /home/thor/playbook
ansible-playbook -i inventory playbook.yml
```

Day 88: Ansible Blockinfile Module

Task Description:

The Nautilus DevOps team wants to install and set up a simple `httpd` web server on all app servers in `Stratos DC`. Additionally, they want to deploy a sample web page for now using Ansible only. Therefore, write the required playbook to complete this task. Find more details about the task below.

We already have an `inventory` file under `/home/thor/ansible` directory on `jump host`. Create a `playbook.yml` under `/home/thor/ansible` directory on `jump host` itself.

- Using the playbook, install `httpd` web server on all app servers. Additionally, make sure its service should up and running.
- Using `blockinfile` Ansible module add some content in `/var/www/html/index.html` file. Below is the content:
  ```
  Welcome to XfusionCorp!
  This is Nautilus sample file, created using Ansible!
  Please do not modify this file manually!
  ```
- The `/var/www/html/index.html` file's user and group `owner` should be `apache` on all app servers.
- The `/var/www/html/index.html` file's permissions should be `0644` on all app servers.

Solution:

Create the playbook

```
cd /home/thor/ansible
vi playbook.yml
```

Add the Content:

```
---
- name: Install and configure httpd web server on all app servers
  hosts: app_servers
  become: yes
  tasks:
    - name: Install httpd package
      package:
        name: httpd
        state: present

    - name: Ensure /var/www/html directory exists
      file:
        path: /var/www/html
```

```
      state: directory
      mode: '0755'

  - name: Ensure index.html exists
    file:
      path: /var/www/html/index.html
      state: touch
      mode: '0644'

  - name: Ensure httpd service is running and enabled
    service:
      name: httpd
      state: started
      enabled: yes

  - name: Add content to index.html using blockinfile
    blockinfile:
      path: /var/www/html/index.html
      block: |
        Welcome to XfusionCorp!

        This is  Nautilus sample file, created using Ansible!

        Please do not modify this file manually!

  - name: Set ownership of index.html
    file:
      path: /var/www/html/index.html
      owner: apache
      group: apache

  - name: Set permissions of index.html
    file:
      path: /var/www/html/index.html
      mode: '0644'
```

Run Ansible Playbook:

```
cd /home/thor/ansible
ansible-playbook -i inventory playbook.yml
```

Day 89: Ansible Manage Services

Task Description:

Developers are looking for dependencies to be installed and run on Nautilus app servers in Stratos DC. They have shared some requirements with the DevOps team. Because we are now managing packages installation and services management using Ansible, some playbooks need to be created and tested. As per details mentioned below please complete the task:

a. On `jump host` create an Ansible playbook `/home/thor/ansible/playbook.yml` and configure it to install `httpd` on all app servers.
b. After installation make sure to start and enable `httpd` service on all app servers.
c. The inventory `/home/thor/ansible/inventory` is already there on `jump host`.
d. Make sure user `thor` should be able to run the playbook on `jump host`.

Solution:

Create the playbook

```
cd /home/thor/ansible
vi playbook.yml
```

Add the Content:

```
---
- name: Install and configure httpd on all app servers
  hosts: app_servers
  become: yes
  tasks:
    - name: Install httpd package
      package:
        name: httpd
        state: present

    - name: Start and enable httpd service
      service:
        name: httpd
        state: started
        enabled: yes
```

Run Ansible Playbook:

```
cd /home/thor/ansible
ansible-playbook -i inventory playbook.yml
```

Day 90: Managing ACLs Using Ansible

Task Description:

There are some files that need to be created on all app servers in `Stratos DC`. The Nautilus DevOps team want these files to be owned by user `root` only however, they also want the app specific user to have a set of permissions on these files. All tasks must be done using Ansible only, so they need to create a playbook. Below you can find more information about the task.

Create a playbook named `playbook.yml` under `/home/thor/ansible` directory on `jump host`, an `inventory` file is already present under `/home/thor/ansible` directory on `Jump Server` itself.s

1. Create an empty file `blog.txt` under `/opt/security/` directory on app server 1. Set some `acl` properties for this file. Using `acl` provide `read '(r)'` permissions to `group tony` (i.e `entity` is `tony` and `etype` is `group`).
2. Create an empty file `story.txt` under `/opt/security/` directory on app server 2. Set some `acl` properties for this file. Using `acl` provide `read + write '(rw)'` permissions to `user steve` (i.e `entity` is `steve` and `etype` is `user`).
3. Create an empty file `media.txt` under `/opt/security/` on app server 3. Set some `acl` properties for this file. Using `acl` provide `read + write '(rw)'` permissions to `group banner` (i.e `entity` is `banner` and `etype` is `group`).

Solution:

Create the playbook

```
cd /home/thor/ansible
vi playbook.yml
```

Add the Content:

```
---
- name: Manage security files and set ACLs on app servers
  hosts: app_servers
  become: yes
  tasks:
    - name: Create /opt/security directory if not exists
      file:
        path: /opt/security
        state: directory
        owner: root
        group: root
        mode: '0755'

    - name: Create blog.txt on app server 1 and set ACL for group tony
      file:
        path: /opt/security/blog.txt
        state: touch
        owner: root
        group: root
        mode: '0644'
      when: inventory_hostname == "stapp01"

    - name: Set ACL read permission for group tony on blog.txt
      acl:
        path: /opt/security/blog.txt
        entity: tony
        etype: group
        permissions: r
        state: present
      when: inventory_hostname == "stapp01"

    - name: Create story.txt on app server 2 and set ACL for user steve
      file:
        path: /opt/security/story.txt
        state: touch
        owner: root
        group: root
        mode: '0644'
      when: inventory_hostname == "stapp02"

    - name: Set ACL read+write permission for user steve on story.txt
      acl:
        path: /opt/security/story.txt
```

```
        entity: steve
        etype: user
        permissions: rw
        state: present
      when: inventory_hostname == "stapp02"

    - name: Create media.txt on app server 3 and set ACL for group banner
      file:
        path: /opt/security/media.txt
        state: touch
        owner: root
        group: root
        mode: '0644'
      when: inventory_hostname == "stapp03"

    - name: Set ACL read+write permission for group banner on media.txt
      acl:
        path: /opt/security/media.txt
        entity: banner
        etype: group
        permissions: rw
        state: present
      when: inventory_hostname == "stapp03"
```

Run Ansible Playbook:

```
cd /home/thor/ansible
ansible-playbook -i inventory playbook.yml
```

Day 91: Ansible Lineinfile Module

Task Description:

The Nautilus DevOps team want to install and set up a simple `httpd` web server on all app servers in `Stratos DC`. They also want to deploy a sample web page using Ansible. Therefore, write the required playbook to complete this task as per details mentioned below.

We already have an `inventory` file under `/home/thor/ansible` directory on `jump host`. Write a playbook `playbook.yml` under `/home/thor/ansible` directory on `jump host` itself. Using the playbook perform below given tasks:

1. Install `httpd` web server on all app servers, and make sure its service is up and running.
2. Create a file `/var/www/html/index.html` with content:

```
This is a Nautilus sample file, created using Ansible!
```

3. Using `lineinfile` Ansible module add some more content in `/var/www/html/index.html` file. Below is the content:

```
Welcome to xFusionCorp Industries!
```

Also make sure this new line is added at the top of the file.

4. The `/var/www/html/index.html` file's user and group `owner` should be `apache` on all app servers.
5. The `/var/www/html/index.html` file's permissions should be `0655` on all app servers.

Solution:

Create the playbook

```
cd /home/thor/ansible
vi playbook.yml
```

Add the Content:

```
---
- name: Install and configure httpd web server
  hosts: app_servers
  become: yes
  tasks:

    - name: Install httpd package
      yum:
        name: httpd
        state: present

    - name: Ensure httpd service is enabled and running
      service:
        name: httpd
        state: started
        enabled: yes

    - name: Create sample index.html file
      copy:
        dest: /var/www/html/index.html
        content: "This is a Nautilus sample file, created using Ansible!\n"
        owner: apache
        group: apache
        mode: '0655'

    - name: Add additional content at the top of index.html
      lineinfile:
        path: /var/www/html/index.html
        line: "Welcome to xFusionCorp Industries!"
        insertbefore: BOF
        owner: apache
        group: apache
        mode: '0655'
```

Run Ansible Playbook:

```
cd /home/thor/ansible
ansible-playbook -i inventory playbook.yml
```

Day 92: Managing Jinja2 Templates Using Ansible

Task Description:

One of the Nautilus DevOps team members is working on to develop a role for `httpd` installation and configuration. Work is almost completed, however there is a requirement to add a jinja2 template for `index.html` file. Additionally, the relevant task needs to be added inside the role. The inventory file `~/ansible/inventory` is already present on `jump host` that can be used. Complete the task as per details mentioned below:

a. Update `~/ansible/playbook.yml` playbook to run the `httpd` role on `App Server 2`.

b. Create a jinja2 template `index.html.j2` under `/home/thor/ansible/role/httpd/templates/` directory and add a line `This file was created using Ansible on <respective server>` (for example `This file was created using Ansible on stapp01` in case of `App Server 1`). Also please make sure not to hard code the server name inside the template. Instead, use `inventory_hostname` variable to fetch the correct value.

c. Add a task inside `/home/thor/ansible/role/httpd/tasks/main.yml` to copy this template on `App Server 2` under `/var/www/html/index.html`. Also make sure that `/var/www/html/index.html` file's permissions are `0644`.

d. The user/group owner of `/var/www/html/index.html` file must be respective sudo user of the server (for example `tony` in case of `stapp01`).

Solution:

Update playbook file

```
cd /home/thor/ansible/
vi playbook.yml
```

We need it to run only on **App Server 2**.
Add the hostname in the hosts

```
---
- hosts: stapp02
  become: yes
  become_user: root
  roles:
    - role: httpd
```

Create the Jinja2 template

```
cd /home/thor/ansible/role/httpd/templates/
vi index.html.j2
```

Add the content

```
This file was created using Ansible on {{ inventory_hostname }}
```

That ensures it dynamically inserts the server name.

Add the task to copy the template

```
cd /home/thor/ansible/role/httpd/tasks/
vi main.yml
```

Add a new task at the end of the file:

```
- name: Deploy custom index.html from template
  template:
    src: index.html.j2
    dest: /var/www/html/index.html
    owner: "{{ ansible_user }}"
    group: "{{ ansible_user }}"
    mode: '0644'
```

The owner and group are set to {{ ansible_user }} which resolves to the respective sudo user.

Run Playbook

```
cd ~/ansible
ansible-playbook -i inventory playbook.yml
```

Day 93: Using Ansible Conditionals

Task Description:

The Nautilus DevOps team had a discussion about, how they can train different team members to use Ansible for different automation tasks. There are numerous ways to perform a particular task using Ansible, but we want to utilize each aspect that Ansible offers. The team wants to utilise Ansible's conditionals to perform the following task:

An `inventory` file is already placed under `/home/thor/ansible` directory on `jump host`, with all the `Stratos DC app servers` included.

Create a playbook `/home/thor/ansible/playbook.yml` and make sure to use Ansible's `when` conditionals statements to perform the below given tasks.

1. Copy `blog.txt` file present under `/usr/src/itadmin` directory on `jump host` to `App Server 1` under `/opt/itadmin` directory. Its user and group owner must be user `tony` and its permissions must be `0644` .
2. Copy `story.txt` file present under `/usr/src/itadmin` directory on `jump host` to `App Server 2` under `/opt/itadmin` directory. Its user and group owner must be user `steve` and its permissions must be `0644` .
3. Copy `media.txt` file present under `/usr/src/itadmin` directory on `jump host` to `App Server 3` under `/opt/itadmin` directory. Its user and group owner must be user `banner` and its permissions must be `0644`.

`NOTE:` You can use `ansible_nodename` variable from gathered facts with `when` condition. Additionally, please make sure you are running the play for all hosts i.e use `- hosts: all`.

Solution:

Create playbook file

```
cd /home/thor/ansible/
vi playbook.yml
```

Add the content in the file

```
---
- name: Conditional file copy to app servers
  hosts: all
  become: yes
  tasks:
    - name: Copy blog.txt to App Server 1
```

```
    copy:
      src: /usr/src/itadmin/blog.txt
      dest: /opt/itadmin/blog.txt
      owner: tony
      group: tony
      mode: '0644'
    when: ansible_nodename == "stapp01.stratos.xfusioncorp.com"

  - name: Copy story.txt to App Server 2
    copy:
      src: /usr/src/itadmin/story.txt
      dest: /opt/itadmin/story.txt
      owner: steve
      group: steve
      mode: '0644'
    when: ansible_nodename == "stapp02.stratos.xfusioncorp.com"

  - name: Copy media.txt to App Server 3
    copy:
      src: /usr/src/itadmin/media.txt
      dest: /opt/itadmin/media.txt
      owner: banner
      group: banner
      mode: '0644'
    when: ansible_nodename == "stapp01.stratos.xfusioncorp.com"
```

If we want to check the ansible nodename

```
ansible all -i inventory -m setup -a 'filter=ansible_nodename'
```

Run Playbook

```
cd ~/ansible
ansible-playbook -i inventory playbook.yml
```

Terraform

Day 94: Create VPC Using Terraform

Task Description:

The Nautilus DevOps team is strategizing the migration of a portion of their infrastructure to the AWS cloud. Recognizing the scale of this undertaking, they have opted to approach the migration in incremental steps rather than as a single massive transition. To achieve this, they have segmented large tasks into smaller, more manageable units. This granular approach enables the team to execute the migration in gradual phases, ensuring smoother implementation and minimizing disruption to ongoing operations. By breaking down the migration into smaller tasks, the Nautilus DevOps team can systematically progress through each stage, allowing for better control, risk mitigation, and optimization of resources throughout the migration process.

Create a VPC named `devops-vpc` in region `us-east-1` with any `IPv4` CIDR block through terraform.

The Terraform working directory is `/home/bob/terraform`. Create the `main.tf` file (do not create a different `.tf` file) to accomplish this task.

Solution:

We already have the AWS provider configured in a separate **provider.tf** file under **/home/bob/terraform**

Open Terminal in vscode and Create **main.tf** file

```
cd /home/bob/terraform/
touch main.tf
```

Add the content in **main.tf** file

```
resource "aws_vpc" "devops-vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "devops-vpc"
  }
}
```

Save the file then execute the terraform commands:
Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```

Day 95: Create Security Group Using Terraform

Task Description:

The Nautilus DevOps team is strategizing the migration of a portion of their infrastructure to the AWS cloud. Recognizing the scale of this undertaking, they have opted to approach the migration in incremental steps rather than as a single massive transition. To achieve this, they have segmented large tasks into smaller, more manageable units. This granular approach enables the team to execute the migration in gradual phases, ensuring smoother implementation and minimizing disruption to ongoing operations. By breaking down the migration into smaller tasks, the Nautilus DevOps team can systematically progress through each stage, allowing for better control, risk mitigation, and optimization of resources throughout the migration process.

Use Terraform to create a security group under the default VPC with the following requirements:

1) The name of the security group must be `datacenter-sg`.

2) The description must be `Security group for Nautilus App Servers`.

3) Add an inbound rule of type `HTTP`, with a port range of `80`, and source CIDR range `0.0.0.0/0`.

4) Add another inbound rule of type `SSH`, with a port range of `22`, and source CIDR range `0.0.0.0/0`.

Ensure that the security group is created in the us-east-1 region using Terraform. The Terraform working directory is `/home/bob/terraform`. Create the `main.tf` file (do not create a different `.tf` file) to accomplish this task.

Solution:

We already have the AWS provider configured in a separate **provider.tf** file under **/home/bob/terraform**

Open Terminal in vscode and Create **main.tf** file

```
cd /home/bob/terraform/
touch main.tf
```

Add the content in **main.tf** file

```
data "aws_vpc" "default" {
  default = true
}

resource "aws_security_group" "datacenter-sg" {
  name        = "datacenter-sg"
  description = "Security group for Nautilus App Servers"
  vpc_id      = data.aws_vpc.default.id

  ingress {
    description = "Allow HTTP traffic"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "Allow SSH traffic"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "datacenter-sg"
  }
}
```

Save the file then execute the terraform commands:
Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```

Day 96: Create EC2 Instance Using Terraform

Task Description:

The Nautilus DevOps team is strategizing the migration of a portion of their infrastructure to the AWS cloud. Recognizing the scale of this undertaking, they have opted to approach the migration in incremental steps rather than as a single massive transition. To achieve this, they have segmented large tasks into smaller, more manageable units.

For this task, create an EC2 instance using `Terraform` with the following requirements:

- The name of the instance must be `datacenter-ec2`.
- Use the `Amazon Linux ami-0c101f26f147fa7fd` to launch this instance.
- The Instance type must be `t2.micro`.
- Create a new RSA key named `datacenter-kp`.
- Attach the default (available by default) security group.

The Terraform working directory is `/home/bob/terraform`. Create the `main.tf` file (do not create a different `.tf` file) to provision the instance.

Solution:

We already have the AWS provider configured in a separate **provider.tf** file under **/home/bob/terraform**

Open Terminal in vscode and Create **main.tf** file

```
cd /home/bob/terraform/
touch main.tf
```

Add the content in **main.tf** file

```
data "aws_vpc" "default" {
  default = true
}

data "aws_security_group" "default_sg" {
  vpc_id = data.aws_vpc.default.id
  name   = "default"
}

resource "tls_private_key" "datacenter_kp" {
```

```
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "local_file" "datacenter_kp_file" {
  content  = tls_private_key.datacenter_kp.private_key_pem
  filename = "datacenter-kp.pem"
  file_permission = "0600"
}

resource "aws_key_pair" "datacenter_kp" {
  key_name   = "datacenter-kp"
  public_key = tls_private_key.datacenter_kp.public_key_openssh
}

resource "aws_instance" "datacenter_ec2" {
  ami           = "ami-0c101f26f147fa7fd"
  instance_type = "t2.micro"
  key_name = aws_key_pair.datacenter_kp.key_name
  vpc_security_group_ids = [data.aws_security_group.default_sg.id]
  tags = {
    Name = "datacenter-ec2"
  }
}
```

Save the file then execute the terraform commands:
Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```

Day 97: Create IAM Policy Using Terraform

Task Description:

When establishing infrastructure on the AWS cloud, Identity and Access Management (IAM) is among the first and most critical services to configure. IAM facilitates the creation and management of user accounts, groups, roles, policies, and other access controls. The Nautilus DevOps team is currently in the process of configuring these resources and has outlined the following requirements.

Create an IAM policy named `iampolicy_mariyam` in `us-east-1` region using Terraform. It must allow read-only access to the EC2 console, i.e., this policy must allow users to view all instances, AMIs, and snapshots in the Amazon EC2 console.

The Terraform working directory is `/home/bob/terraform`. Create the `main.tf` file (do not create a different `.tf` file) to accomplish this task.

Solution:

We already have the AWS provider configured in a separate **provider.tf** file under **/home/bob/terraform**

Open Terminal in vscode and Create **main.tf** file

```
cd /home/bob/terraform/
touch main.tf
```

Add the content in **main.tf** file

```
resource "aws_iam_policy" "iampolicy_mariyam" {
  name        = "iampolicy_mariyam"
  description = "Read-only access to the EC2 console including instances,
AMIs, and snapshots"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect   = "Allow"
        Action   = [
          "ec2:Describe*"
        ]
        Resource = "*"
```

```
      }
    ]
  })
}
```

Save the file then execute the terraform commands:
Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```

Day 98: Launch EC2 in Private VPC Subnet Using Terraform

Task Description:

The Nautilus DevOps team is expanding their AWS infrastructure and requires the setup of a private Virtual Private Cloud (VPC) along with a subnet. This VPC and subnet configuration will ensure that resources deployed within them remain isolated from external networks and can only communicate within the VPC. Additionally, the team needs to provision an EC2 instance under the newly created private VPC. This instance should be accessible only from within the VPC, allowing for secure communication and resource management within the AWS environment.

1. Create a VPC named `nautilus-priv-vpc` with the CIDR block `10.0.0.0/16`.
2. Create a subnet named `nautilus-priv-subnet` inside the VPC with the CIDR block `10.0.1.0/24` and `auto-assign` IP option must not be `enabled`.
3. Create an EC2 instance named `nautilus-priv-ec2` inside the subnet and instance type must be `t2.micro`.
4. Ensure the security group of the EC2 instance allows access only from within the VPC's CIDR block.
5. Create the `main.tf` file (do not create a separate `.tf` file) to provision the VPC, subnet and EC2 instance.
6. Use `variables.tf` file with the following variable names:
    - `KKE_VPC_CIDR` for the VPC CIDR block.
    - `KKE_SUBNET_CIDR` for the subnet CIDR block.
7. Use the `outputs.tf` file with the following variable names:
    - `KKE_vpc_name` for the name of the VPC.
    - `KKE_subnet_name` for the name of the subnet.
    - `KKE_ec2_private` for the name of the EC2 instance.

Solution:

We already have the AWS provider configured in a separate **provider.tf** file under **/home/bob/terraform**

Open Terminal in vscode and Create **main.tf** file

```
cd /home/bob/terraform/
touch main.tf
touch variables.tf
touch outputs.tf
```

Add the content in **main.tf** file

```
resource "aws_vpc" "nautilus_priv_vpc" {
  cidr_block           = var.KKE_VPC_CIDR
  tags = {
    Name = "nautilus-priv-vpc"
  }
}

resource "aws_subnet" "nautilus_priv_subnet" {
  vpc_id            = aws_vpc.nautilus_priv_vpc.id
  cidr_block        = var.KKE_SUBNET_CIDR
  map_public_ip_on_launch = false
  tags = {
    Name = "nautilus-priv-subnet"
  }
}

resource "aws_security_group" "nautilus_priv_sg" {
  name        = "nautilus-priv-sg"
  description = "Allow access only from within VPC"
  vpc_id      = aws_vpc.nautilus_priv_vpc.id

  ingress {
    from_port   = 0
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = [var.KKE_VPC_CIDR]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = [var.KKE_VPC_CIDR]
  }

  tags = {
    Name = "nautilus-priv-sg"
  }
}

resource "aws_instance" "nautilus_priv_ec2" {
```

```
  ami           = "ami-0c101f26f147fa7fd"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.nautilus_priv_subnet.id
  vpc_security_group_ids = [aws_security_group.nautilus_priv_sg.id]

  tags = {
    Name = "nautilus-priv-ec2"
  }
}
```

Add the content in **variables.tf** file

```
variable "KKE_VPC_CIDR" {
  description = "CIDR block for the VPC"
  default     = "10.0.0.0/16"
}

variable "KKE_SUBNET_CIDR" {
  description = "CIDR block for the subnet"
  default     = "10.0.1.0/24"
}
```

Add the Content on **outputs.tf** file

```
output "KKE_vpc_name" {
  value = aws_vpc.nautilus_priv_vpc.tags["Name"]
}

output "KKE_subnet_name" {
  value = aws_subnet.nautilus_priv_subnet.tags["Name"]
}

output "KKE_ec2_private" {
  value = aws_instance.nautilus_priv_ec2.tags["Name"]
}
```

Save the file then execute the terraform commands:

Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```

Day 99: Attach IAM Policy for DynamoDB Access Using Terraform

Task Description:

The DevOps team has been tasked with creating a secure DynamoDB table and enforcing fine-grained access control using IAM. This setup will allow secure and restricted access to the table from trusted AWS services only.

As a member of the Nautilus DevOps Team, your task is to perform the following using Terraform:

1. **Create a DynamoDB Table:** Create a table named `xfusion-table` with minimal configuration.
2. **Create an IAM Role:** Create an IAM role named `xfusion-role` that will be allowed to access the table.
3. **Create an IAM Policy:** Create a policy named `xfusion-readonly-policy` that should grant read-only access (GetItem, Scan, Query) to the specific DynamoDB table and attach it to the role.
4. Create the `main.tf` file (do not create a separate `.tf` file) to provision the table, role, and policy.
5. Create the `variables.tf` file with the following variables:
    - `KKE_TABLE_NAME`: name of the DynamoDB table
    - `KKE_ROLE_NAME`: name of the IAM role
    - `KKE_POLICY_NAME`: name of the IAM policy
6. Create the `outputs.tf` file with the following outputs:
    - `kke_dynamodb_table`: name of the DynamoDB table
    - `kke_iam_role_name`: name of the IAM role
    - `kke_iam_policy_name`: name of the IAM policy
7. Define the actual values for these variables in the `terraform.tfvars` file.
8. Ensure that the `IAM policy` allows only read access and restricts it to the specific `DynamoDB table` created.

Solution:

We already have the AWS provider configured in a separate **provider.tf** file under **/home/bob/terraform**

Open Terminal in vscode and Create **main.tf** file

```
cd /home/bob/terraform/
touch main.tf
touch variables.tf
touch terraform.tfvars
touch outputs.tf
```

Add the content in **main.tf** file

```
resource "aws_dynamodb_table" "xfusion_table" {
  name           = var.KKE_TABLE_NAME
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "id"

  attribute {
    name = "id"
    type = "S"
  }

  tags = {
    Name = var.KKE_TABLE_NAME
  }
}

resource "aws_iam_role" "xfusion_role" {
  name = var.KKE_ROLE_NAME

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Service = [
            "lambda.amazonaws.com",
            "ec2.amazonaws.com",
            "dynamodb.amazonaws.com"
          ]
        }
        Action = "sts:AssumeRole"
      }
    ]
  })
}
```

```
resource "aws_iam_policy" "xfusion_readonly_policy" {
  name        = var.KKE_POLICY_NAME
  description = "Read-only access to the specific DynamoDB table"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "dynamodb:GetItem",
          "dynamodb:Scan",
          "dynamodb:Query"
        ]
        Resource = aws_dynamodb_table.xfusion_table.arn
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "xfusion_policy_attachment" {
  role       = aws_iam_role.xfusion_role.name
  policy_arn = aws_iam_policy.xfusion_readonly_policy.arn
}
```

Add the content in **variables.tf** file

```
variable "KKE_TABLE_NAME" {
  description = "Name of the DynamoDB table"
  type       = string
}

variable "KKE_ROLE_NAME" {
  description = "Name of the IAM role"
  type       = string
}

variable "KKE_POLICY_NAME" {
  description = "Name of the IAM policy"
  type       = string
}
```

Add the Content on **terraform.tfvars** file

```
KKE_TABLE_NAME = "xfusion-table"
KKE_ROLE_NAME  = "xfusion-role"
KKE_POLICY_NAME = "xfusion-readonly-policy"
```

Add the Content on **outputs.tf** file

```
output "kke_dynamodb_table" {
  description = "Name of the DynamoDB table"
  value       = aws_dynamodb_table.xfusion_table.name
}

output "kke_iam_role_name" {
  description = "Name of the IAM role"
  value       = aws_iam_role.xfusion_role.name
}

output "kke_iam_policy_name" {
  description = "Name of the IAM policy"
  value       = aws_iam_policy.xfusion_readonly_policy.name
}
```

Save the file then execute the terraform commands:

Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```

Day 100: Create and Configure Alarm Using CloudWatch Using Terraform

Task Description:

The Nautilus DevOps team has been tasked with setting up an EC2 instance for their application. To ensure the application performs optimally, they also need to create a CloudWatch alarm to monitor the instance's CPU utilization. The alarm should trigger if the CPU utilization exceeds 90% for one consecutive 5-minute period. To send notifications, use the SNS topic named `nautilus-sns-topic`, which is already created.

1. **Launch EC2 Instance:** Create an EC2 instance named `nautilus-ec2` using any appropriate Ubuntu AMI (you can use AMI `ami-0c02fb55956c7d316`).
2. **Create CloudWatch Alarm:** Create a CloudWatch alarm named `nautilus-alarm` with the following specifications:
    - Statistic: Average
    - Metric: CPU Utilization
    - Threshold: >= 90% for 1 consecutive 5-minute period
    - Alarm Actions: Send a notification to the `nautilus-sns-topic` SNS topic.
3. Update the `main.tf` file to create a EC2 Instance and CloudWatch Alarm.
4. Create an `outputs.tf` file to output the following values:

- `KKE_instance_name` for the EC2 instance name.
- `KKE_alarm_name` for the CloudWatch alarm name.

Solution:

We already have the AWS provider file (provider.tf) and Main configuration file (main.tf) in the **/home/bob/terraform**

Open Terminal in vscode and Create **outputs.tf** file

```
cd /home/bob/terraform/
touch outputs.tf
```

Add the Content on **outputs.tf** file

```
output "KKE_instance_name" {
  value = aws_instance.nautilus_ec2.tags["Name"]
}

output "KKE_alarm_name" {
```

```
  value = aws_cloudwatch_metric_alarm.nautilus_alarm.alarm_name
}
```

Update the **main.tf** file

```
resource "aws_sns_topic" "sns_topic" {
  name = "nautilus-sns-topic"
}

resource "aws_instance" "nautilus_ec2" {
  ami           = "ami-0c02fb55956c7d316"
  instance_type = "t2.micro"
  tags = {
    Name = "nautilus-ec2"
  }
}

resource "aws_cloudwatch_metric_alarm" "nautilus_alarm" {
  alarm_name          = "nautilus-alarm"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = 1
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = 300
  statistic           = "Average"
  threshold           = 90
  alarm_description   = "Triggers if CPU Utilization >= 90% for one 5-minute
period"
  alarm_actions       = [aws_sns_topic.sns_topic.arn]

  dimensions = {
    InstanceId = aws_instance.nautilus_ec2.id
  }
}
```

Save the file then execute the terraform commands:

Initialize Terraform

```
terraform init
```

Validate configuration:

```
terraform validate
```

Review the plan:

```
terraform plan
```

Apply the configuration:

```
terraform apply
```