**OSS All Experiments**

1. **FTP - Installation, configuration and its use.**

FTP, which stands for File Transfer Protocol, is a standard network protocol used to transfer files from one host to another over a TCP-based network, such as the internet. It is a client-server protocol, where the client initiates a connection to the server to request files or to send files to the server.

Installing an FTP server involves setting up the software on a server machine to enable it to respond to FTP requests from clients. The server software manages the file transfers and provides access to the files and directories on the server to authorized users.

**Step1: Update package index using following command**

**sudo apt update**

**Step2: Install ftp server by the command  
 sudo apt install vsftpd**

**Step3: Checking the status of the ftp server  
 sudo service vsftpd status**

**{The server should be running}**

**Step4: Configure ftp server  
 sudo nano /etc/vsftpd.conf**

**{In the file}**

**Make local\_enable=YES  
 write\_enable=YES**

**chroot\_local\_user=YES**

**user\_sub\_token=$USER**

**local\_root=/home/$USER/ftp**

**pasv\_min\_port=10000  
 pasv\_max\_port=10100**

**userlist\_enable=YES**

**userlist\_file=/etc/vsftpd.userlist**

**userlist\_deny=NO**

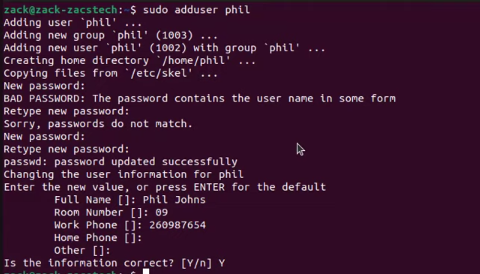
**Add all of the lines**

**Press ctrl+X → Y → Enter**

**Step5: Configure firewall and open these ports**

**sudo ufw allow from any to any port 20,21,10000:10100 proto tcp**

**Step6: add user  
sudo adduser phil**

****

**Step7: create ftp folder for the user in the home directory  
Sudo mkdir /home/phil/ftp**

**Step8: run the command to configure permissions  
sudo chown nobody:nogroup /home/phil/ftp**

**Step9: permission for the home directory**

**sudo chmod a-w /home/phil/ftp**

**Step10: make upload folder for the user  
Sudo mkdir /home/phil/ftp/upload**

**Step11: upload ownership for the user  
sudo chown phil:phil /home/phil/ftp/upload**

**Step12: make demo file into the upload folder  
echo “My ftp server” | sudo tee /home/phil/ftp/upload/demo.txt**

**Step13: check the ftp permission of the file by the command  
sudo ls -la /home/phil/ftp**

**Step14: login and access the ftp server  
Echo “phil” | sudo tee -a /etc/vsftpd.userlist**

**Step15: restarting ftp server  
Sudo systemctl restart vsftpd  
  
Step16: checking IP address of the machine to check whether user has the access to ftp server  
Ifconfig —> then copy first IP address  
  
Step17: install filezilla and input IP address(host), password and username and do quickconnect   
You will get an upload folder and inside that demo.txt file**

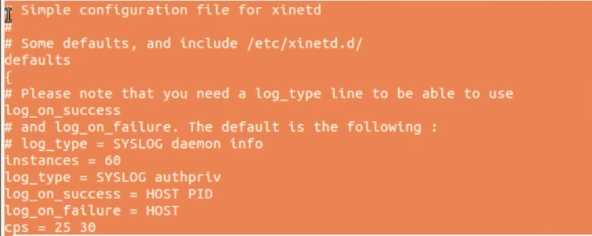
**sudo service vsftpd stop**

**Telnet configuration and use**

**Telnet is a network protocol used for connecting to remote systems through a command-line interface. It can be used to configure remote devices such as routers, switches, servers, etc. Telnet can also be used to test if a TCP port on a remote system is open or not.  
  
Telnet is not a secure protocol because the Telnet session between server and client is unencrypted. You can use it for testing the connectivity to TCP ports. However, for connecting to the remote system, it is recommended to use SSH.**

**Step1: install telnet using the command  
sudo apt-get install xinetd telnetd**

**Step2: configuring the inetd.conf file and adding permission  
{add this line}  
telnet stream tcp nowait telnetd /user/sbin/tcpd /user/sbin/in.telnetd  
  
Step3: editing xinetd.conf file**

**Sudo gedit /etc/xinetd.conf  
**

**Step4: Start telnet server  
sudo /etc/init.d/xinetd restart**

**Step5: Testing telnet server**

**Both PC should be connected to same network  
Note down the IP address of HOST machine (PC1) using {ifconfig command}**

**PC1: ifconfig  
PC2: telent <IP Address>**

**PC1: make a file and check it by command {ls}**

**PC2: type {ls} command and you will see same file appearing on PC2.**

**NFS(Network File System) Configuration**

**NFS allows a system to share directories and files with others over a network. By using NFS, users and programs can access files on remote systems almost as if they were local files.**

**Some of the most notable benefits that NFS can provide are:**

* **Local workstations use less disk space because commonly used data can be stored on a single machine and still remain accessible to others over the network.**
* **There is no need for users to have separate home directories on every network machine. Home directories could be set up on the NFS server and made available throughout the network.**
* **Storage devices such as floppy disks, CDROM drives, and USB Thumb drives can be used by other machines on the network. This may reduce the number of removable media drives throughout the network.**

**1. Git offline: on local machine with multiple user**

**If you're working with Git on a local machine with multiple users and you want to set up a**

**repository for collaborative development offline, you can follow these steps. In this example, let's consider two users, `user1` and `user2**

**Step1: Create new folder  
Step2: Initialize the git repo {git init}**

**Step3: Configure user information**

**# For user1**

**git config user.name "User1 Name"**

**git config user.email "**[**user1@example.com**](mailto:user1@example.com)**"**

**# For user2**

**git config user.name "User2 Name"**

**git config user.email "**[**user2@example.com**](mailto:user2@example.com)**"**

**Step4: Create a new branch for each user to work on:**

**```bash**

**# User1 creates and switches to a new branch**

**git checkout -b user1\_branch**

**# User2 creates and switches to a new branch**

**git checkout -b user2\_branch**

**Step5: Work on respective branches**

**#User1**

**git add .**

**git commit -m "User1's changes"**

**#User2**

**git add .**

**git commit -m "User2's changes"**

**Step6: Switch between branches**

**# User1 switches to User2's branch**

**git checkout user2\_branch**

**Step7: Merge changes**

**#User1 merges changes to the main branch**

**git checkout main**

**git merge user1\_branch**

**#User2 merges changes to main branch**

**git checkout main**

**git merge user2\_branch**

**- Ensure proper communication between users to avoid conflicts and coordinate development**

**efforts.**

**- Remember that this approach is for collaborative development on a local machine. If you plan**

**to work across different machines or want a backup of your repository, consider using a remote**

**repository (e.g., GitHub, GitLab) for more robust version control.**

**These steps provide a basic workflow for collaborative Git development on a local machine.**

**Adjustments might be necessary based on the specific requirements and collaboration patterns**

**of your project.**

**2. Demonstrate the use/features of online Bug Tracking/Issue Tracking "BugZilla".**

**Bugzilla is a** [**web**](https://en.wikipedia.org/wiki/World_Wide_Web)**-based general-purpose** [**bug tracking system**](https://en.wikipedia.org/wiki/Bug_tracking_system) **and** [**testing tool**](https://en.wikipedia.org/wiki/Test_automation) **originally developed and used by the** [**Mozilla**](https://en.wikipedia.org/wiki/Mozilla) **project.**

**Step1: getting bugzilla file from github**

**{in terminal full command} wget https://gist.githubusercontent.com/anefzaoui/73f3b397abe4e5b1bef994a3c2a9f480/r aw/79f802c06db66955262f1f1c05dbdb10f9d626af/bugzilla\_installation\_ubuntu.sh**

**Step2: permission to file  
Chmod +x bugzilla\_installation\_ubuntu.sh**

**Step3: installation of bugzilla**

**./bugzilla\_installation\_ubuntu.sh**

**Step4: Input necessary information and you bugzilla instance will be created**

**Later you can visit the website**

**6. Creating of RPM Package{requires Red Hat distro eg. fedora}**

**RPM is both the package manager and the package format used by many linux distributions such as fedora, red hat and centos, to manage and distribute software in binary form.**

**The root of an rpm build environment tree is the rpmbuild directory which contains 6 subdirectories: BUILD, BUILDROOT, RPMS, SOURCES, SPECS, SRPMS**

**Step1: To create rpm packages get source code**

**8. Creating of Debian Packages**

**Vedant Joshi Answer maybe it is not the best way: create rpm package and then convert it into debian based package**

**Step1: install alien package to convert rpm to debian**

**Sudo apt install alien**

**Step2: Get any .rpm package from internet**

**Step3: convert the rpm package into debian**

**sudo alien -k -d <Package\_name>**

**{-k for generating same version  
-d for generating debian package}**

**11. Create package create rpm package**

Create a folder in vscode

1. Pura idhar se chaap aur nahi hua to steps follow kr
2. Debian package alag se download kr aur wo downloaded sir ko dikha de
3. Khalas
4. https://betterprogramming.pub/how-to-create-a-basic-debian-package-927be001ad80

**10. Demonstrate the use/features of project management tool: ‘SONAR’ for managing project**

**1. Install OpenJDK 11**

**SSH to your Ubuntu server as a non-root user with sudo access.**

**Install OpenJDK 11.**

**$ sudo apt-get install openjdk-11-jdk -y**

**------------------------------------------------------------------------------------------**

**2. Install and Configure PostgreSQL**

**Add the PostgreSQL repository.**

**$ sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb\_release -cs`-pgdg main" /etc/apt/sources.list.d/pgdg.list'**

**Add the PostgreSQL signing key.**

**$ wget -q https://www.postgresql.org/media/keys... -O - | sudo apt-key add -**

**Install PostgreSQL.**

**$ sudo apt install postgresql postgresql-contrib -y**

**Enable the database server to start automatically on reboot.**

**$ sudo systemctl enable postgresql**

**Start the database server.**

**$ sudo systemctl start postgresql**

**Change the default PostgreSQL password.**

**$ sudo passwd postgres**

**Switch to the postgres user.**

**$ su - postgres**

**Create a user named sonar.**

**$ createuser sonar**

**Log in to PostgreSQL.**

**$ psql**

**Set a password for the sonar user. Use a strong password in place of my\_strong\_password.**

**ALTER USER sonar WITH ENCRYPTED password 'my\_strong\_password';**

**Create a sonarqube database and set the owner to sonar.**

**CREATE DATABASE sonarqube OWNER sonar;**

**Grant all the privileges on the sonarqube database to the sonar user.**

**GRANT ALL PRIVILEGES ON DATABASE sonarqube to sonar;**

**Exit PostgreSQL.**

**\q**

**Return to your non-root sudo user account.**

**$ exit**

**-----------------------------------------------------------------------------------------**

**3. Download and Install SonarQube**

**Install the zip utility, which is needed to unzip the SonarQube files.**

**$ sudo apt-get install zip -y**

**Locate the latest download URL from the SonarQube official download page.**

**Download the SonarQube distribution files.**

**$ sudo wget https://binaries.sonarsource.com/Dist...**

**Unzip the downloaded file.**

**sudo unzip sonarqube-9.6.1.59531.zip**

**Move the unzipped files to /opt/sonarqube directory**

**sudo mv sonarqube-9.6.1.59531 sonarqube**

**sudo mv sonarqube /opt/**

**4. Add SonarQube Group and User**

**Create a dedicated user and group for SonarQube, which can not run as the root user.**

**Create a sonar group.**

**$ sudo groupadd sonar**

**Create a sonar user and set /opt/sonarqube as the home directory.**

**$ sudo useradd -d /opt/sonarqube -g sonar sonar**

**Grant the sonar user access to the /opt/sonarqube directory.**

**$ sudo chown sonar:sonar /opt/sonarqube -R**

**-----------------------------------------------------------------------------------------**

**5. Configure SonarQube**

**Edit the SonarQube configuration file.**

**$ sudo nano /opt/sonarqube/conf/sonar.properties**

**Find the following lines:**

**#sonar.jdbc.username=**

**#sonar.jdbc.password=**

**Uncomment the lines, and add the database user and password you created in Step 2.**

**sonar.jdbc.username=sonar**

**sonar.jdbc.password=my\_strong\_password**

**Below those two lines, add the sonar.jdbc.url.**

**sonar.jdbc.url=jdbc:postgresql://localhost:5432/sonarqube**

**Save and exit the file.**

**Edit the sonar script file.**

**$ sudo nano /opt/sonarqube/bin/linux-x86-64/sonar.sh**

**About 50 lines down, locate this line:**

**#RUN\_AS\_USER=**

**Uncomment the line and change it to:**

**RUN\_AS\_USER=sonar**

**Save and exit the file.**

**-----------------------------------------------------------------------------------------**

**6. Setup Systemd service**

**Create a systemd service file to start SonarQube at system boot.**

**$ sudo nano /etc/systemd/system/sonar.service**

**Paste the following lines to the file.**

**[Unit]**

**Description=SonarQube service**

**After=syslog.target network.target**

**[Service]**

**Type=forking**

**ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start**

**ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop**

**User=sonar**

**Group=sonar**

**Restart=always**

**LimitNOFILE=65536**

**LimitNPROC=4096**

**[Install]**

**WantedBy=multi-user.target**

**Save and exit the file.**

**Enable the SonarQube service to run at system startup.**

**$ sudo systemctl enable sonar**

**Start the SonarQube service.**

**$ sudo systemctl start sonar**

**Check the service status.**

**$ sudo systemctl status sonar**

**-----------------------------------------------------------------------------------------**

**7. Modify Kernel System Limits**

**SonarQube uses Elasticsearch to store its indices in an MMap FS directory. It requires some changes to the system defaults.**

**Edit the sysctl configuration file.**

**$ sudo nano /etc/sysctl.conf**

**Add the following lines.**

**vm.max\_map\_count=262144**

**fs.file-max=65536**

**ulimit -n 65536**

**ulimit -u 4096**

**Save and exit the file.**

**Reboot the system to apply the changes.**

**$ sudo reboot**

**-----------------------------------------------------------------------------------------**

**8. Access SonarQube Web Interface**

**Access SonarQube in a web browser at your server's IP address on port 9000. For example:**

**http://IP:9000**

**Log in with username admin and password admin. SonarQube will prompt you to change your password.**

**Flask app in docker image**

**#### Type the commands in main terminal not of VS code terminal ####**

**Step1: pip install flask also install docker**

**Step2: make a file index.py**

from flask import Flask

helloworld = Flask(\_\_name\_\_)

@helloworld.route("/")

def run():

return "Hello World"

if \_\_name\_\_ == "\_\_main\_\_":

helloworld.run(host="0.0.0.0", port=int("3000"), debug=True)

**Step3: make a docker file**

FROM python:3-alpine3.15

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

EXPOSE 3000

CMD python ./index.py

**Step4: install flask and make requirements.txt**

flask==3.0.0

**Step5: make a container of docker**

**Sudo docker build -t flask\_app:latest .**

**Step6: run the docker file**

**Sudo docker run -d -p 3000:3000 flask\_app:latest**

**{YOU can check the site running on localhost:3000}**

**Step7: change the container name   
sudo docker -t flask\_app <dockerhub\_username>/flask-app**

**Step8: push the image**

**sudo docker push <dockerhub\_username>/flask-app:latest**

**19. Communication between docker containers.**

**24. Flask application creation using docker and creating its image.**

**https://youtu.be/PXo3AAquPy0?si=uw8dtip32\_FZulEN**

The docker containers can communicate with one another by two ways:

1. Having shared directory
2. Through the networking, just by starting both the containers they will be able to communicate with one another.

Flask is a lightweight and versatile web framework for Python that is commonly used t o build web applications and APIs. Building a RESTful API using Flask involves creating a web application that adheres to the principles of Representational State Transfer (REST). REST is an architectural style for designing networked applications, and it uses a stateless communication model based on standard HTTP methods (GET, POST, PUT, DELETE) and resources identified by URIs (Uniform Resource Identifiers).

Step1: Create folder name container\_communicate

Step2: Within that folder create folder name container1.

Step3: In container1 create Dockerfile and app.py file.

Step4: app.py file contains following code.

from flask import Flask

from flask\_restful import Resource, Api

app = Flask(\_\_name\_\_)

api = Api(app)

class SendMessage(Resource):

def get(self):

return {"Message": "Message from container 1"}

api.add\_resource(SendMessage, '/')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', debug=True)

Now, let me explain the code:

1. **Import Statements:**
   * from flask import Flask: Imports the Flask class, which is used to create a Flask web application.
   * from flask\_restful import Resource, Api: Imports the Resource class and Api class from the flask\_restful extension. These classes are used for creating RESTful resources.
2. **Create Flask App and API:**
   * app = Flask(\_\_name\_\_): Creates an instance of the Flask class. The \_\_name\_\_ argument is a special Python variable that represents the name of the current module.
   * api = Api(app): Creates an instance of the Api class, passing the Flask app instance as an argument. This sets up the Flask-RESTful extension.
3. **Define a Resource:**
   * class SendMessage(Resource):: Defines a class called SendMessage that inherits from the Resource class. In Flask-RESTful, resources are used to define endpoints and handle HTTP methods (GET, POST, etc.).
   * def get(self):: Defines a method within the SendMessage class to handle HTTP GET requests. In this case, it returns a simple JSON response with a message.
4. **Add Resource to API:**
   * api.add\_resource(SendMessage, '/'): Associates the SendMessage resource with a specific URL endpoint ('/'). This means that when a request is made to the root URL, the SendMessage resource will handle it.
5. **Run the Application:**
   * if \_\_name\_\_ == '\_\_main\_\_':: This conditional statement ensures that the Flask app is only run if the script is executed directly (not if it's imported as a module).
   * app.run(host='0.0.0.0', debug=True): Starts the development server. The host='0.0.0.0' parameter makes the server externally accessible, and debug=True enables debugging mode.

To test this code, you can run the script and access the API endpoint using a tool like curl or a web browser. For example, if you run the script and open<http://localhost:5000> in your browser, you should see the JSON response: {"Message": "Message from container 1"}.

Step5: The content of Dockerfile will be as follows.

# Use the latest Alpine Linux as the base image

FROM alpine:latest

# Set the working directory inside the container

WORKDIR /app

# Copy the Python script to the container

COPY ./app.py /app/

# Install Python 3, pip3, and create a virtual environment

RUN apk add --no-cache python3 py3-pip \

&& python3 -m venv /venv

# Set the virtual environment as the active environment

ENV PATH="/venv/bin:$PATH"

# Install Flask within the virtual environment

RUN pip3 install Flask

RUN pip3 install flask-restful

# Command to run when the container starts

CMD ["python3", "/app/app.py"]

1. FROM alpine:latest: Specifies that the base image for the Docker image is Alpine Linux, a lightweight and minimal Linux distribution.
2. RUN apk update: Updates the package index within the Alpine Linux image.
3. RUN apk add python3 py3-pip: Installs Python 3 and pip3, the package manager for Python 3, using the Alpine package manager (apk).
4. RUN pip3 install flask flask-restful: Installs the Flask web framework and Flask-RESTful extension using pip3.
5. RUN mkdir /app: Creates a directory named /app in the container.
6. COPY ./app.py /app/app.py: Copies the app.py file from the host machine to the /app directory in the container. This assumes that there is an app.py file in the same directory as the Dockerfile.
7. CMD python3 /app/app.py: Sets the default command to run when the container starts. It specifies that the Python script app.py located in the /app directory should be executed.

Step6: Go inside the container1 and open the terminal.

Step7: Now let’s build our docker container.

sudo docker build . -t image1 —--------------------------------------------> (Building docker file. We are at the location where the docker file is, hence ‘.’ 1.0.0 is specifying the version.)

Step8: docker images —-------------------> (It will show your image named container1)

Step9: sudo docker run -it image1 —-----------> (-d: Run the container in detached mode, meaning it runs in the background.

Step10: Go to the localhost address provided. You will see the message.

Step11: Now create another container. For this you can just pull the pre-existing image from docker demon.

Sudo docker run -it alpine:latest

Step12: wget -q -o - 172.17.0.4:5000 —---------------------> After hitting this command you will receive the message sent from first container. But while hitting this command your first container must be running.

Wget -qO - 172.17.0.4:5000

Step13: You can check the network list. For that you can just come out of this by entering exit.

sudo docker network list

Step14: To stop the running container of image1, you can hit the following command

sudo docker stop container-id

Done boyzzz….Flask wala done………………………………………………………

**20. Demonstrate the use/features of Project Management tool for managing projects.**

Some well-known tools are as follows:

1. Redmine: Project management tool and issue tracking web application. Also it is open source with GNU public license.
2. Taiga: Open source project management tool for agile development. Follows same licence.
3. Jira: Commercial. Follows GNU license.
4. Trello: Free but premium version is also available.
5. Clickup: Free but premium version is also available.
6. Github: Distributed version control system. Free but paid is also available.
7. Bitbucket: Source code management platform. Free but paid version is also available.

Step1: Pull the LAMP(Linux,apache,mysql,php) stack container

docker pull bitnami/lamp:latest

**22.**