**To connect to an Amazon EC2 instance using Cygwin**, you can use the OpenSSH client that is included with Cygwin. Here's a step-by-step guide:

\*\*1. Install Cygwin:\*\*

- Download and run the Cygwin installer from the official website: [Cygwin](https://www.cygwin.com/).

- During the installation, make sure to select the "openssh" package from the Net category.

\*\*2. Generate SSH Key Pair:\*\*

- Open Cygwin terminal.

- Run the following command to generate an SSH key pair if you don't have one:

```bash

ssh-keygen -t rsa -b 2048

```

- Press Enter to accept the default file location and set a passphrase if needed.

\*\*3. Copy SSH Public Key to EC2 Instance:\*\*

- Navigate to the EC2 Dashboard on the AWS Management Console.

- Select your instance and click on "Connect."

- Copy the public key (usually located in `~/.ssh/id\_rsa.pub` unless you specified a different path).

\*\*4. Configure SSH Agent (Optional):\*\*

- To avoid entering your passphrase every time you connect, you can add your private key to the SSH agent:

```bash

eval $(ssh-agent -s)

ssh-add ~/.ssh/id\_rsa

```

\*\*5. Connect to EC2 Instance:\*\*

- Open Cygwin terminal.

- Use the following command to connect to your EC2 instance:

```bash

**ssh -i /path/to/private/key.pem ec2-user@your-instance-ip**

```

Replace `/path/to/private/key.pem` with the path to your private key file and `your-instance-ip` with the public IP address or DNS name of your EC2 instance.

\*\*Note:\*\*

- Ensure that your security group allows incoming SSH traffic (port 22) from your local IP address.

- Replace `ec2-user` with the appropriate username for your AMI (Amazon Machine Image). Common usernames are `ec2-user`, `ubuntu`, or `admin` depending on the AMI.

By following these steps, you should be able to connect to your EC2 instance using Cygwin and SSH.

///////////////////////////////////////////////////////////////////////////////////////////////////

**Docker install**

Sure, here are step-by-step commands to install Docker on an Ubuntu 22.04 instance:

\*\*1. Update Package List:\*\*

```bash

sudo apt update

```

\*\*2. Install Required Dependencies:\*\*

```bash

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

```

\*\*3. Add Docker GPG Key:\*\*

```bash

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

```

\*\*4. Add Docker Repository:\*\*

```bash

echo "deb [signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

```

\*\*5. Update Package List Again:\*\*

```bash

sudo apt update

```

\*\*6. Install Docker Engine:\*\*

```bash

sudo apt install -y docker-ce docker-ce-cli containerd.io

```

\*\*7. Add Your User to the Docker Group:\*\*

```bash

sudo usermod -aG docker $USER

```

\*Note: To apply the group membership, you may need to log out and log back in, or run `newgrp docker`.\*

\*\*8. Verify Docker Installation:\*\*

```bash

docker --version

```

\*\*9. Run a Test Container:\*\*

```bash

docker run hello-world

```

This should download a small test image and run a container that prints a "Hello from Docker!" message.

Now, Docker is successfully installed on your Ubuntu 22.04 EC2 instance. You can start using Docker to manage containers on your machine. Remember that you may need to use `sudo` or re-login to apply the group changes if you encounter permission issues.

To check the status of Docker images on your system, you can use the following Docker command:

```bash

**docker images**

```

This command lists all the Docker images that are currently available on your system. The output will include information such as the image ID, repository, tag, and size.

If you are using a more recent version of Docker, you can also use the following command to achieve the same result:

```bash

docker image ls

```

Both commands provide a snapshot of the Docker images present on your machine, allowing you to see what images are available locally.

//////////////////////////////////////////////////////////////////////////////////////////////////

**To connect to an Amazon EC2 instance using PuTTY** on a Windows machine, you'll need the private key (.pem file) associated with your EC2 instance. Here are the steps:

### Step 1: Convert PEM to PPK (PuTTY Private Key)

1. If you don't have PuTTYgen installed, download it from the [PuTTY Download Page](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html).

2. Open PuTTYgen.

3. Click on "Load" and select your .pem file.

4. Click "Save private key" to save the private key in .ppk format.

### Step 2: Connect with PuTTY

1. Download and install PuTTY from the [PuTTY Download Page](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html).

2. Open PuTTY.

3. In the "Session" category, enter your EC2 instance's public IP address or DNS hostname in the "Host Name (or IP address)" field.

4. In the "Connection" category, expand "SSH" and select "Auth." Browse and select the .ppk private key file you generated.

5. Optionally, you can save the session for future use by entering a name in the "Saved Sessions" field and clicking "Save."

6. Click "Open" to initiate the connection.

### Step 3: Log in to the EC2 Instance

- Once the PuTTY terminal opens, log in using the appropriate username. For Amazon Linux or Amazon AMIs, the default username is usually `ec2-user`. For Ubuntu instances, it is `ubuntu`. For other AMIs, check the AMI documentation for the correct username.

- Copy the username and paste it in the PuTTY terminal, then press Enter.

- You are now logged into your EC2 instance via PuTTY.

Remember that the security group associated with your EC2 instance must allow SSH traffic (port 22) from your local machine. Also, use the proper private key associated with your EC2 instance during PuTTY configuration.

/////////////////////////////////////////////////////////////////////////////////////////////////

1 ls

2 sudo apt update

3 sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

4 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

5 echo "deb [signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

6 sudo apt update

7 sudo apt install -y docker-ce docker-ce-cli containerd.io

8 sudo usermod -aG docker $USER

9 docker --version

10 docker run hello-world

11 docker images

12 sudo docker images

13 sudo docker run hello-world

14 sudo docker images

15 sudo docker images ls

////////////////////////////////////////////////////////

1 sudo apt update

2 sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

3 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

4 echo "deb [signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

5 sudo apt update

6 sudo apt install -y docker-ce docker-ce-cli containerd.io

7 sudo usermod -aG docker $USER

8 docker --version

9 C:\Users\jains

10 docker run -d -p 80:80 --name my-nginx nginx

11 sudo docker run -d -p 80:80 --name my-nginx nginx

12 sudo docker ps -a

13 sudo docker stop ea80c85de1c0

14 sudo docker rm ea80c85de1c0

15 sudo docker ps -a

16 history

///////////////////////////////////////////////

1 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash

3 sudo apt update

4 . ~/.nvm/nvm.sh

5 nvm install --lts

6 node -e "console.log('Running Node.js ' + process.version)"

7 ls

8 mkdir myapp

9 cd myapp/

10 vim server.js

11 ls

12

13 node --version

14 npm --version

15 npm install -g nodemon

16 ls

17 nodemon server.js

25 cd ..

26 npx create-react-app my-react-app

27 cd my-react-app/

28 ls

29 npm start

30 cd ..

31 cd myapp

39 cd ..

40 sudo apt update

41 sudo apt install -y docker.io

42 sudo docker ps -a

43 cd my-react-app/

44 nano Dockerfile

45 docker build -t my-react-app .

46 sudo docker build -t my-react-app .

47 docker run -p 3000:3000 my-react-app

48 sudo docker run -p 3000:3000 my-react-app

49 history

Folder remove commond -rm-rf foldername/

Sudo rm

/////////////////////////////////////////

**To install a React app inside a Docker container on an EC2 Ubuntu instance, you can follow these general steps:**

### Step 1: Connect to Your EC2 Instance

Use SSH to connect to your EC2 instance:

```bash

ssh -i "your-key.pem" ubuntu@your-ec2-public-ip

```

### Step 2: Install Docker

If Docker is not already installed on your EC2 instance, install it by following the steps mentioned in a previous response:

```bash

sudo apt update

sudo apt install -y docker.io

```

### Step 3: Create a React App

Navigate to a directory where you want to create your React app:

```bash

mkdir my-react-app

cd my-react-app

```

Create a new React app using `create-react-app`:

```bash

npx create-react-app .

```

### Step 4: Create a Dockerfile

Create a `Dockerfile` in the root of your React app directory:

```Dockerfile

# Use an official Node runtime as a parent image

FROM node:14-alpine

# Set the working directory to /app

WORKDIR /app

# Copy package.json and package-lock.json to the working directory

COPY package\*.json ./

# Install app dependencies

RUN npm install

# Copy the current directory contents into the container at /app

COPY . .

# Make port 3000 available to the world outside this container

EXPOSE 3000

# Define environment variable

ENV REACT\_APP\_API\_URL=https://api.example.com

# Run app when the container launches

CMD ["npm", "start"]

```

This `Dockerfile` uses the official Node.js Alpine image, sets up the working directory, installs dependencies, exposes port 3000, and runs the React app.

### Step 5: Build the Docker Image

Build the Docker image:

```bash

docker build -t my-react-app .

```

### Step 6: Run the Docker Container

Run the Docker container:

```bash

docker run -p 3000:3000 my-react-app

```

The `-p 3000:3000` flag maps port 3000 on your local machine to port 3000 in the Docker container.

### Step 7: Access the React App in the Browser

Open your web browser and navigate to your EC2 instance's public IP address followed by `:3000`:

```

http://your-ec2-public-ip:3000

```

You should see your React app running.

That's it! You've successfully installed and run a React app inside a Docker container on your EC2 instance.

////////////////////////////////////////

///////////////////////////////////////////////////

EC2 commands

1 sudo apt update

2 sudo apt install -y nginx

25 sudo nano index.html

31 sudo find / -name "index.html"

32 sudo chown -R ubuntu:ubuntu /var/www/html/

33 sudo service nginx restart

34 history

Local pc commands

jains@HP /cygdrive/c/Users/jains/Downloads/project

$ scp -i "docker-key.pem" -r "/cygdrive/c/Users/jains/Downloads/project/\*" ubuntu@ec2-18-212-73-213.compute-1.amazonaws.com:/var/www/html/

1. **Install nodejs in aws ec2 instance :-**
2. Connect to your Linux instance as ec2-user using SSH.
3. Install node version manager (nvm) by typing the following at the command line.

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash

1. Activate nvm by typing the following at the command line

. ~/.nvm/nvm.sh

1. Use nvm to install the latest LTS version of Node.js by typing the following at the command line

nvm install --lts

1. Test that Node.js is installed and running correctly by typing the following at the command line

node -e "console.log('Running Node.js ' + process.version)"