**Traditional Data Center Infrastructure**

1. **Components**:
   * **Servers**: Physical machines hosting applications.
   * **Routers**: Direct network traffic between different networks.
   * **Switches**: Connect devices within the same network, facilitating communication.
   * **Disks**: Storage devices for data.
2. **Operation**:
   * **Physical Hosting**: All applications and services are hosted on physical machines in a data center. This requires extensive capacity planning to ensure adequate resources for varying user loads over time. For instance, if user capacity grows from 500 in January to 5,000 by September, additional resources must be provisioned.
3. **Infrastructure Engineers**: Responsible for deploying applications on physical hosts based on planned capacity. This often involves purchasing hardware, configuring systems, and maintaining them.

**Virtualized Infrastructure**

1. **Components**:
   * **Hypervisor**: Software that creates and manages virtual machines (VMs). It allows multiple VMs to run on a single physical server, improving resource utilization.
     + **Types**:
       - **VMware**: A leading virtualization platform known for robust features and support.
       - **Oracle VirtualBox**: A free and open-source hypervisor suitable for local development and testing.
       - **KVM (Kernel-based Virtual Machine)**: A Linux kernel module that allows the kernel to function as a hypervisor.
       - **HVM (Hardware Virtual Machine)**: A virtualization technology that allows multiple OS instances to run simultaneously on hardware.
2. **Benefits**:
   * **Resource Efficiency**: Virtualization enables running multiple operating systems on the same hardware, reducing the need for physical servers.
   * **Scalability**: Resources can be allocated dynamically based on demand, making it easier to scale applications as user numbers increase.
   * **Minimal Overhead**: Cloud providers typically offer minimal costs per hour for virtual resources, allowing businesses to pay only for what they use.

**Cloud Infrastructure**

1. **Cloud Providers**:
   * **AWS, Azure, GCP**: Leading cloud service providers offering Infrastructure as a Service (IaaS), which allows businesses to rent virtualized resources instead of investing in physical hardware.
   * **Benefits**:
     + **Flexibility**: Easily scale resources up or down based on real-time demand.
     + **Cost-Effectiveness**: Pay-as-you-go pricing models reduce costs associated with underutilized resources.
2. **Business Owner Perspective**:
   * **Raman (as the business owner)**: Focuses on leveraging cloud services for efficient resource management, lower upfront investment, and improved agility to adapt to changing business needs.

**Conclusion**

The shift from traditional to virtualized and cloud-based infrastructure represents a significant evolution in how businesses approach IT resource management. With virtualization and cloud computing, organizations can achieve better resource utilization, flexibility, and cost-effectiveness, adapting quickly to user demands and operational requirements.

**Example Use Case**

* **Scenario**: A startup launches an application in January expecting 500 users but anticipates rapid growth to 5,000 by September.
* **Traditional Approach**: They might overprovision physical servers to accommodate peak demand, leading to excess capacity and wasted resources.
* **Virtualized/Cloud Approach**: They deploy the application on a cloud platform, starting with minimal resources and scaling up as user demand increases, optimizing costs and efficiency.

This flexibility and scalability offered by virtualization and cloud solutions are crucial for modern businesses aiming for rapid growth and adaptation in a dynamic environment.

**Lab Guide: Creating an Ubuntu Linux Instance on AWS**

**Prerequisites**

1. **AWS Account**: Ensure you have an AWS account. If not, sign up at [aws.amazon.com](https://aws.amazon.com/).
2. **IAM User**: Create an IAM user with the necessary permissions to launch EC2 instances.
3. **AWS CLI**: Optionally, install the AWS CLI on your local machine for command-line access to AWS services.

**Step 1: IAM User and Policy Setup**

1. **Log in to AWS Management Console**.
2. Navigate to **IAM (Identity and Access Management)**.
3. Click on **Users** and then **Add user**.
4. Enter a username and select **Programmatic access** and **AWS Management Console access**.
5. Set a custom password or allow AWS to auto-generate one.
6. Click **Next: Permissions**.
7. Choose **Attach existing policies directly**.
8. Select the following policies (or create a custom policy with similar permissions):
   * AmazonEC2FullAccess (for EC2 instance management)
   * IAMReadOnlyAccess (optional for viewing IAM details)
9. Click **Next: Tags** and optionally add tags.
10. Review and create the user. Save the **Access Key ID** and **Secret Access Key** securely.

**Step 2: Creating a Security Group**

1. Navigate to the **EC2 Dashboard**.
2. On the left pane, select **Security Groups**.
3. Click **Create security group**.
4. Fill in the following details:
   * **Group Name**: MySecurityGroup
   * **Description**: Allow SSH access
   * **VPC**: Choose the default VPC or your custom VPC.
5. Under **Inbound rules**, click **Add rule**:
   * **Type**: SSH
   * **Protocol**: TCP
   * **Port Range**: 22
   * **Source**: Anywhere (0.0.0.0/0) for testing or restrict to your IP for security.
6. Click **Create security group**.

**Step 3: Launching an Ubuntu EC2 Instance**

1. Navigate to the **EC2 Dashboard**.
2. Click on **Instances** in the left pane, then click **Launch instance**.
3. Configure the following settings:
   * **Name your instance**: MyUbuntuInstance
   * **Application and OS Images (Amazon Machine Image)**: Search for Ubuntu and select an appropriate AMI (e.g., Ubuntu Server 22.04 LTS).
   * **Instance Type**: Select t2.micro (eligible for the free tier).
4. Click **Next: Configure Instance**:
   * Leave the defaults or adjust as necessary (like adding a role if needed).
5. Click **Next: Add Storage**:
   * Default is typically 8 GiB; adjust if needed.
6. Click **Next: Add Tags**:
   * Add tags as needed (e.g., Key: Name, Value: MyUbuntuInstance).
7. Click **Next: Configure Security Group**:
   * Select the security group you created earlier (MySecurityGroup).
8. Click **Review and Launch**. Review your settings, and then click **Launch**.
9. When prompted, create a new key pair or select an existing one:
   * If creating a new key pair, download the key file (e.g., my-key.pem) and save it securely.
10. Click **Launch Instances**.

**Step 4: Connecting to Your Instance**

1. **SSH Access**:
   * Ensure you have SSH installed on your local machine (Linux or macOS comes with it; Windows users can use PuTTY or Windows Subsystem for Linux).
2. Set permissions on your key file:

bash

Copy code

chmod 400 my-key.pem

1. Get the public IP address of your instance from the **Instances** page in the EC2 dashboard.
2. Use SSH to connect to your instance:

bash

Copy code

ssh -i "my-key.pem" ubuntu@<your\_instance\_public\_ip>

**Step 5: Post-Login Configuration**

Once logged in to your instance, you can perform initial configuration:

1. **Update Package Index**:

bash

Copy code

sudo apt update

1. **Install Common Packages**:

bash

Copy code

sudo apt install -y vim git curl

1. **Configure Your Application**: If you're hosting an application, configure your application and database servers as needed.

**Additional Notes**

* **Access Keys and Secret Access Keys**: Keep these secure. You can generate temporary access tokens for programmatic access using the AWS CLI or SDKs.
* **Management Console**: Use the AWS Management Console for visual management and monitoring of your resources.
* **AWS CLI**: After installing the AWS CLI, you can configure it using:

bash

Copy code

aws configure

Enter your Access Key ID, Secret Access Key, region, and output format.

**Cleanup**

When you're done testing, terminate your instance to avoid incurring charges:

1. Go to **Instances** in the EC2 Dashboard.
2. Select your instance and click **Actions** → **Instance State** → **Terminate**.

This guide provides a comprehensive approach to launching and managing an Ubuntu EC2 instance on AWS, covering everything from user access to instance security. Adjust the steps as necessary to fit your specific use case or environment!

**Lab Guide: Managing AWS EC2 Instances Using AWS CLI on Ubuntu**

**Prerequisites**

* An AWS account.
* SSH access to an Ubuntu server.

**Step 1: Log in to Your Ubuntu Server**

1. Open your terminal.
2. Use the following command to SSH into your Ubuntu server. Replace <your\_instance\_public\_ip> with your instance’s public IP address:

bash

Copy code

ssh -i "your-key.pem" ubuntu@<your\_instance\_public\_ip>

**Step 2: Install the AWS CLI**

1. **Update the Package Index**:

bash

Copy code

sudo apt update

1. **Install Required Packages**: Install unzip and wget which will be used to download and extract the AWS CLI installer.

bash

Copy code

sudo apt install unzip wget -y

1. **Download the AWS CLI Installer**: Download the latest AWS CLI v2.

bash

Copy code

wget "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -O awscliv2.zip

1. **Unzip the Installer**:

bash

Copy code

unzip awscliv2.zip

1. **Run the Installer**:

bash

Copy code

sudo ./aws/install

1. **Verify Installation**: After installation, verify that the AWS CLI is installed correctly:

bash

Copy code

aws --version

**Step 3: Configure AWS CLI Authentication**

1. **Run the Configuration Command**:

bash

Copy code

aws configure

1. **Enter Your AWS Credentials**: You will be prompted to enter the following:
   * **AWS Access Key ID**: Your access key.
   * **AWS Secret Access Key**: Your secret key.
   * **Default region name**: (e.g., us-east-1, us-west-2, etc.).
   * **Default output format**: (e.g., json, text, table).

**Step 4: Managing EC2 Instances**

**Viewing EC2 Instances**

1. **Describe Instances**: To list all EC2 instances and their details, use:

bash

Copy code

aws ec2 describe-instances --query "Reservations[\*].Instances[\*].{InstanceID:InstanceId,State:State.Name,Type:InstanceType,PublicIP:PublicIpAddress,PrivateIP:PrivateIpAddress,LaunchTime:LaunchTime}" --output table

1. **Filter Instances by Instance ID**: To find a specific instance by its ID:

bash

Copy code

aws ec2 describe-instances | grep -i instanceid

**Creating a New EC2 Instance**

1. **Run Instance Command**: To create a new EC2 instance (replace the ami-xxxx with the appropriate AMI ID):

bash

Copy code

aws ec2 run-instances --image-id ami-0866a3c8686eaeeba --instance-type t2.micro --key-name raman-synechron\_8th --count 1 --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=MyUbuntuServer}]'

**Stopping and Terminating Instances**

1. **Stop an Instance**: To stop an EC2 instance, use the following command with your instance ID:

bash

Copy code

aws ec2 stop-instances --instance-ids <your\_instance\_id>

1. **Terminate an Instance**: To permanently delete an EC2 instance, use:

bash

Copy code

aws ec2 terminate-instances --instance-ids <your\_instance\_id>

**Step 5: Working with IAM Users**

**Create a New IAM User**

1. **Create an IAM User**:

bash

Copy code

aws iam create-user --user-name MyUser

1. **Attach Policies to the User**: To grant the new user full access to S3:

bash

Copy code

aws iam attach-user-policy --user-name MyUser --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess

1. **List All Users**: To verify the user has been created:

bash

Copy code

aws iam list-users

**Additional CLI Commands**

1. **List S3 Buckets**:

bash

Copy code

aws s3 ls

1. **Get Parameters from SSM** (Optional): If you want to retrieve the latest Ubuntu AMI from SSM Parameter Store:

bash

Copy code

aws ssm get-parameters --names /aws/service/canonical/ubuntu/server/20.04/stable/current/amd64/hvm/ebs-gp2 --query 'Parameters[0].Value'

**Cleanup**

When you are finished testing, remember to terminate any EC2 instances you have created to avoid unnecessary charges:

bash

Copy code

aws ec2 terminate-instances --instance-ids <your\_instance\_id>

**Conclusion**

This lab guide covers everything from setting up the AWS CLI on an Ubuntu server to managing EC2 instances and IAM users. You can expand upon these commands and explore further AWS services using the CLI as needed.

Make sure to always monitor your AWS usage to keep track of resources and avoid unexpected charges.

**Creating an EC2 Instance Role**

Here’s an example of how to create an IAM role for an EC2 instance that needs access to S3.

1. **Create Role**: Follow the steps above to create a new role.
2. **Select Trusted Entity**: Choose AWS service, then select EC2.
3. **Attach Permissions Policy**: Attach the AmazonS3ReadOnlyAccess policy (or a custom policy if you need specific permissions).
4. **Name Your Role**: Name the role something like EC2S3AccessRole and finish creating it.

**Launch an EC2 Instance with the Role**

1. Launch a new EC2 instance.
2. In the **IAM role** dropdown, select EC2S3AccessRole.
3. Complete the instance setup.

The EC2 instance will now have permission to access S3 without needing to manage credentials manually.