lab

The AWS Cloud using EC2 Instance (PEM)

- **Launch Instance**:
  - Give a **Name** (e.g., "AWS_PEM-22BD1A0529").
  - Choose **Ubuntu** and **64-bit x86 architecture**.
  - Select **t2.micro** instance type.
  - **Create a new key pair**, download the **.pem file** and save it securely.
  - **Enable Allow HTTP/HTTPS traffic**.
  - Use default **8 GB Storage**.
  - Set **Number of instances: 1**.
- **Connect to Instance**:
  - Ensure **Instance State is running** and **Status Check is 2/2 checks passed**.
  - Select instance, click **Connect**.
  - Copy the **SSH command**.
  - Open PowerShell, navigate to where the .pem file is saved, **paste the SSH command**, and press Enter. Type "yes" if prompted.
- **Terminate Instance**:
  - Select the EC2 instance, click **Instance State**, and choose **Terminate Instance**.

The AWS Cloud using EC2 Instance (PUTTY)

- **Start AWS Lab**.
- Go to **EC2** under Compute services.
- **Launch Instance**:
  - Give a **Name** (e.g., "AWS_PUTTY-22BD1A0529").
  - Choose **Ubuntu**.
  - Select **t2.micro** instance type.
  - **Create a new key pair**, download **.ppk & .pem file** and save securely.
  - **Enable Allow HTTP/HTTPS traffic**.
  - Use default **8 GB Storage**.
  - Set **Number of instances: 2**.
- **Accessing with Putty**:
  - **Download Putty application**.
  - Ensure **Instance State is running** and **Status Check is 2/2 checks passed**.
  - Select instance, click **Connect**, copy the instance connection string.
  - Open Putty, **paste the string into Hostname**.
  - Provide **.ppk file path** in SSH -> Auth -> Credentials.
  - Click **Open** and then **Accept** to launch VM.
- **Accessing using EC2 Direct**:
  - Select **EC2 Instance Connect** option and click **Connect button**.
- **Terminate Instance**:
  - Select the EC2 instance, click **Instance State**, and choose **Terminate Instance**.
- **End the lab session**.

Creating EC2 Instance with Amazon Linux AMI

- **Launch the Instance**:
  - Enter a **Name** (e.g., "529-linuxmachine").
  - Choose **Amazon Linux AMI** and **x86 architecture**.

    - Select **t2.micro** instance type.
    - Select an **existing keypair** (e.g., "newpem.pem").
    - **Allow HTTP/HTTPS** and **SSH** in Network Settings.
    - Use default **8GiB Storage**.
    - Set **Number of Instances: 1**.
- **Connect to Instance**:
    - Ensure **Instance State is running** and **Status Check is 2/2 checks passed**.
    - Select instance, click **Connect**.
    - **Copy the SSH command**.
    - Open terminal, navigate to .pem file location, **paste SSH command**, and press Enter. Type "yes" if prompted.
- **Install Apache**:
    - Run `sudo yum update -y`.
    - Run `sudo yum install httpd`.
    - Run `sudo systemctl enable httpd`.
    - Run `sudo systemctl status httpd` to check status.
- **Accessing VM Using Direct Connect**:
    - Select **EC2 Instance Connect** and click **Connect button**.
- **Check Output**: Go to the **public IP address** to see the output.
- **Terminate Instance**:
    - Select the EC2 instance, click **Instance State**, and choose **Terminate Instance**.

Install NGINX and Changing Default Page

- **Launch an Instance with Ubuntu AMI**.
- **Connect Directly with EC2 Instance Connect**.
- **Install Docker**:
    - Update: `sudo apt update`.
    - Install: `sudo apt-get install docker.io`.
    - Check: `docker -version`.
- **Pull NGINX**: `sudo docker pull nginx`.
- **Run NGINX Container**: `docker run -d -p 80:80 -name mynginx nginx`.
- **Check Running Container**: `sudo docker ps`.
- **Verify NGINX**: Go to the **IP address** of the instance.
- **Change Default Page**:
    - Open bash in container: `sudo docker exec -it mynginx /bin/bash`.
    - Navigate to HTML directory: `cd /usr/share/nginx/html`.
    - Edit `index.html`: `nano index.html`.
    - **Change content in `<h1>` tag to roll number** (e.g., "22BD1A0529").
    - Save and exit: `CTRL+O`, `Enter`, `CTRL+X`.
    - Exit container terminal: `exit`.
- **Verify Update**: Go to **public IP address** and check for update.
- **Terminate the instance**.

S3 Bucket Creation & Permissions

- **Enter AWS console** and search for **S3 service**.
- **Click on Create bucket**.
- **Give configurations** and **create bucket**.
- **Upload File**:
    - Click on the created bucket.
    - Click **Upload**, select file, and click **Upload**.
- **Enable Public Access**:

    - Click on the uploaded object and **copy the object URL**.
    - Try accessing the URL (will get error due to permissions).
    - Go to **Permissions** tab of the bucket.
    - Click **Edit** and **disable Block all public access**, then **Save changes**.
    - Click **Edit at Object Ownership**, **enable ACLs**, and **Save changes**.
    - Click **Edit ACL**, **enable permissions for Everyone**, and **Save changes**.
- **Change Object-level Permissions**:
    - Click on the object permissions and **edit the ACL**.
    - **Enable permissions for Everyone** and **Save changes**.
- **Verify Access**: Reload the page, you should now be able to **access the uploaded image**.

Versioning (S3)

- **Create the bucket** (e.g., "kmit--versioning--22bd1a0529").
- Go to **Properties** and **enable bucket versioning**.
- **Upload an object** into the bucket.
- **Delete the object** in the bucket.
- **Delete objects with type "delete marker"** (by showing versions).
- You can now **restore the deleted object**.

Cross-Region Replication (CRR)

- **Create a New Bucket** (source, e.g., "crr-529-src").
- **Change the region**.
- **Create the destination Bucket** (e.g., "crr-529-destination").
- **Enable versioning** on the destination bucket.
- Go to the **Source-bucket**, then **MANAGEMENT** tab, and click **Create replication rule**.
- **Enter name** (e.g., "crrcopy").
- **Choose the destination bucket** (in this account).
- **Choose IAM role** (e.g., "LabRole") and **SAVE the changes**.
- **Upload a file in the source bucket**.
- Go to the **destination-bucket** to verify the object exists.
- **Reverse the process**: Set the destination bucket as source and vice versa, upload an object to the new source to see it reflected in the new destination.

Static Web Hosting (S3)

- **Create a new bucket** with name (e.g., "static--web--host--529") and configure for **public access**, **ACLs enabled**, and **versioning enabled**.
- **Upload the HTML file** as an object into the bucket.
- Go to the bucket **properties** and **enable static web hosting**, specifying the files.
- Select the HTML object and click **Make it public using ACL**.
- **Copy the link of the object**.
- Open the link in a new page to **get the output**.

EBS and CPU in EC2

- **Launch Instance with EBS (Initial)**:
    - **Name**: "EBS-CPU-529".
    - **AMI**: Amazon Linux.
    - **Instance Type**: t2.micro.
    - **Key pair**: ebs_key.pem.
    - **Add new volume: 20 GB** (EBS volume, not encrypted).
- **Change Instance Type**:

    - Observe instance type is t2.micro.
    - **Stop the instance**.
    - Access menu and **change instance type to t2.medium**.
    - Change back to **t2.micro** due to pricing.
    - Connect to instance and run `lsblk` to see 8G root and 20GB EBS.
- **Adding EBS after Creating Instance**:
    - **Create an instance with default configuration** (8GB root storage, Amazon Linux AMI).
    - Connect using .pem key pair (no extra storage observed initially).
    - Go to **Volumes** in AWS console.
    - **Create a volume with 35 GB** in the same zone as the instance.
    - **Attach volume**: Select the new volume, go to **Actions -> Attach volume**, select the instance, set device name as `/dev/sdb`, and click **Attach volume**.
    - Check for 35 GB EBS storage in instance terminal: `sudo su`, `lsblk`.
- **Mount Storage**:
    - `sudo su`, `lsblk`, `lsblk -fs`, `fdisk -l`.
    - `fdisk /dev/xvdb` (n for new, p for primary, default, w to exit).
    - `partprobe`.
    - `lsblk -fs`.
    - `mkfs.xfs /dev/xvdb1` (format with XFS).
    - `mkdir /mnt/529` (create mount point).
    - `mount /dev/xvdb1 /mnt/529`.
    - `lsblk -fs` to check mount points.
- **Persistence**:
    - Change directory to `/mnt/529` and `touch file{1..10}`.
    - Edit `/etc/fstab` using `nano /etc/fstab`.
    - **Add `/dev/xvdb1 /mnt/529 xfs defaults 0 0` to the last line** and save.
- **Unmount and Detach**:
    - `sudo umount /mnt/529`, `sudo umount /dev/xvdb1`.
    - Check with `lsblk -fs`.
    - Go to Volumes page, select volume, **Actions -> Detach volume**.

Creating Snapshot of the Volume

- **Create an instance with default 8GB storage**.
- **Add EBS volume** (e.g., 20 GB).
- Connect to EC2 instance (direct connect or PuTTY), get to root user, list file system, make partition, file system, directory, mount, and persist (referencing EBS document).
- **Create Snapshot**:
    - Go to **Volumes**, select a volume (in N. Virginia).
    - Give **description** and **create snapshot**.
- **Copy Snapshot**:
    - Go to **Snapshot -> Actions -> Copy snapshot**.
    - Choose **destination region** (e.g., us-west-2/Oregon).
- **Create Volume from Snapshot**:
    - Create instance in the **same availability zone** as the target volume.
    - Go to **Snapshots**, select the snapshot, click **Actions -> Create volume from snapshot**.
    - Choose availability zone **same as instance** and **create volume**.
- **Attach Volume to Instance**:
    - Select the new volume, go to **Actions -> Attach volume**, select the instance, and device.
- **Access and Mount**:
    - Connect to instance (e.g., direct connect), run `lsblk`.
    - `sudo su`, `lsblk`, `mkdir /mnt/529`, `mount /dev/xvdb1 /mnt/529`, `cd /mnt/529`.

- Create files to verify access.

Elastic File System (EFS)

```
- referene of ebs
```

- **Create 2 EC2 instances in the same region, different availability zones** (e.g., Instance 1 in US-east-1a, Instance 2 in US-east-1b).
  - **Edit subnet** in network settings for preferred AZ.
  - Give relevant naming for security group.
- **Configure Security Groups**:
  - Select instance, **edit inbound rules** in security group.
  - **Add rule: NFS**, allow traffic from anywhere.
  - Do this for both instances.
- **Create EFS File System**:
  - Search for **Amazon EFS** in AWS Services.
  - Click **Create File System**, give a name, and complete setup.
- **Attach EFS to EC2 Instances**:
  - Connect to EC2 instances using PuTTY (or direct connect).
  - Run `sudo su` and `mkdir efs`.
  - **Install required tools**: `sudo yum install -y amazon-efs-utils`.
  - **Copy mount command** from EFS Attach tab and run on instance (initially might fail due to security group).
  - **Update security groups for the availability zones** to ensure EFS access.
  - **Retry mounting EFS** using the same command; it should now successfully mount.
- **Verify Shared Access**:
  - Create a file in one EC2 instance's EFS mount path (`/efs`).
  - Verify it **instantly reflects and is accessible in the second instance**.

VPC

- **Create a VPC**: Name "MyVPC".
- **Create Subnets**:
  - First subnet: "Web-Server", **CIDR 10.0.1.0/24**.
  - Second subnet: "Db-Server", **CIDR 10.0.2.0/24**.
- **Enable Public IP for Subnet-1 (Web-Server)**:
  - Select subnet, **Actions -> Edit subnet Settings**.
  - Click **Auto-assign** and save.
- **Create Internet Gateway (IGW)**: Name "My-IGW".
- **Attach IGW to VPC**: **Actions -> Attach to VPC**, attach to "MyVPC".
- **Create Route Table**: Name "RouteTable1".
- **Attach Route Table to Subnet1 (Web-Server)**:
  - Select "Web-Server" subnet, click **Save associations**.
  - **Attach Route table to Internet Gateway**: In Actions -> Edit routes, add a route to 0.0.0.0/0 via the IGW.
- **Create Webserver Instance**:
  - Name "Webserver".
  - Key-pair: "webKP30.pem".
  - VPC: "MyVPC", Subnet: "WebServer".
  - Security group: "WebSG 30", **Add rule: HTTP, Source Anywhere**.
- **Create DbServer Instance**:
  - Name "DbServer".
  - Key pair: "dbKP30".
  - VPC: "MyVPC", Subnet: "Db-server".

    - Security group: "Db-SG", **Change type from SSH to MySQL/Aurora**.
- **Open public IP of webserver instance**.

VPC using NAT Gateway

- **Create Bastion Server**:
    - Name: "BastionServer".
    - VPC: "MyVPC", Subnet: **Public (web server subnet)**.
    - Security Group: "Bastion-SG".
    - Connect using Putty key pair.
- **Create DB Server (Private)**:
    - Connect using pem key pair.
    - VPC: "MyVPC", Subnet: **Private (Db server subnet)**.
    - Security Group: "db-sg".
    - SSH Custom: **Give private IP of Bastion server here** (e.g., 10.0.0.143/32).
- **Test Internet from Bastion (Public Subnet)**:
    - Commands like `sudo su`, `yum update -y`, `yum install git` should work.
- **Copy DB Server Key Pair to Bastion (if needed)**:
    - Use **WINSCP** to connect to Bastion server.
    - Copy DB server's .pem file into the Bastion server's Linux machine.
- **Create NAT Gateway**:
    - Create a NAT Gateway **in the public subnet (Web Server)**.
- **Create Route Table for Private Subnet**:
    - Name: "Bastion-RT", VPC: "MyVPC".
    - **Add subnet associations** and connect it to the **private subnet (db server)**.
    - **Edit the routes and add NAT gateway**.
- **Verify Internet Access for DB Server**:
    - Now, internet access should be enabled in the private DB server, accessible only through the bastion server.
    - Test with `yum update -y`, `yum install git`.

Lambda

- **Create Lambda Function**:
    - Search for **Lambda** in AWS dashboard.
    - Click **Create function**.
    - Give function name (e.g., "529_function").
    - Select runtime (e.g., Python 3.9).
    - Select **use an existing role** and set as "LabRole".
    - Click **Create function**.
- **Deploy and Test**:
    - Change `lambda_function.py` and click **Deploy**.
    - In Tests section, **create a new test** and click **Test**.
- **Create Function URL**:
    - Go to **Configuration** section, then **Function URL**.
    - Click **Create new function URL**.
    - Select **"AWS_IAM"** and click **Save**.
    - (Note: Opening link initially gives error).
- **Update Code and Integrate with S3/DynamoDB**:
    - **Copy code from GitHub** and paste into `lambda_function.py`.
    - **Create an S3 bucket** with default config.
    - **Create DynamoDB table**: Name "newtable", partition key "unique".
    - **Deploy the new code**.

    - Create a new test and test the code (errors might still occur).
- **Add S3 Trigger**:
    - Select **S3 as trigger source**.
    - Select the S3 bucket created and add **"All Object create events"**.
    - Tick checkbox and click **Add button**.
- **Verify Trigger**:
    - Go to the S3 bucket and **upload some file**.
    - In the DynamoDB table, click **Explore table items** to see the uploaded file details reflected.

Docker

- **Upload file to GitHub** after creating a repo.
- **Create an instance in AWS**:
    - **AMI**: Amazon Linux.
    - **Instance Type**: t2.micro.
    - **Allow all network rules**.
    - Default storage.
- **Connect to instance using .pem**.
- **Install Git**: `sudo yum install git`.
- **Install Docker**: `sudo yum install docker`.
- **Clone the repo** onto the VM.
- **Create a Dockerfile** in the repository directory with:

```
FROM nginx:alpine
COPY ./portfolio.html /usr/share/nginx/html/index.html
```

  .
- **Start Docker**: `sudo service docker start`.
- **Build Docker Image**: `sudo docker build -t portfolio .`.
- **Run Docker Container**: `sudo docker run -d -p 80:80 portfolio`.
- **Verify Output**: Open the **public IP address of the VM instance** to check the output.

AWS SNS

- **Create SNS Topic**:
    - Search for **SNS** in AWS dashboard.
    - Click **Topics** -> **Create SNS Topic**.
    - **Topic name**: "DemoSNS", **Display name**: "529-Topic".
- **Create Subscription**:
    - Click **Create subscription**.
    - **Protocol**: EMAIL, **Endpoint**: your email (e.g., 22bd1a0529@gmail.com).
    - Click **Create subscription**.
    - **Check your email** and **Click on confirm subscription**.
    - Verify status is **Confirmed** in Topics dashboard.
- **Publish Message**:
    - Go to **Publish message**, enter a raw message, and **Publish the message**.
    - Verify message is received in endpoint email.
- **Integrate with S3 Event Notification**:
    - Go to **S3** and **create a bucket** (e.g., "sns529").
    - Go to bucket **Properties** -> **Event notifications**.
    - **Create event notification**: **Event types** -> **All Object create events**.
    - **Destination**: SNS topic, Specify **SNS topic ARN**.
    - (Initially will get an error when saving).

- **Edit SNS Topic Access Policy**:
  - Go to **SNS topic**, click **Edit**, and open **Access policy**.
  - **Paste the provided JSON code**.
  - **Replace** `SNS-topic-ARN` **and** `arn:aws:s3:*:*:amzn-s3-demo-bucket` **with your actual ARNs**.
  - **Save changes**.
  - Go back to S3 create events and click **Save** (should now succeed).
- **Verify S3 Notification**:
  - **Upload any object into the S3 bucket**.
  - Verify **email notification is received from SNS topic**.

AWS SQS

- **Create SQS Queue**:
  - Search for **SQS** in AWS console.
  - Click **Create queue**.
  - Choose **type as standard**.
  - Keep other configurations as default and click **Create queue**.
- **Send and Receive Messages**:
  - Click on **Send and receive messages**.
  - Type a **message body** and **send the message**.
  - **Poll the message to Receive it**.
- **Create Lambda Function with SQS Trigger**:
  - Go to **AWS Lambda** and **Create a new function using blueprint**.
  - Select **Use an existing role** (e.g., LabRole).
  - **Add a trigger for SQS**.
  - Select the **SQS queue created** and add the trigger.
- **Verify Trigger and Monitoring**:
  - Send a message to the SQS queue (without polling).
  - Check **monitoring tab** in Lambda to see statistics (message should be processed by Lambda).

CloudFront

- **Create an S3 bucket**:
  - **ACL should be enabled to be public**, uncheck Block all access.
  - Create the bucket.
- **Add objects into the S3 bucket**.
- **Enable static web hosting** for the S3 bucket.
- **Create CloudFront Distribution**:
  - Search for **CloudFront**.
  - Select the **S3 bucket** and **create the distribution**.
  - Check the **domain name**.
  - **Enable Web Application Firewall**.
  - Create the distribution.
- **Verify Content Delivery**:
  - **Copy the domain name** of the CloudFront distribution.
  - Access the content using the CloudFront domain name (e.g., `http://d21k6yctdzb8zd.cloudfront.net/demoimg.png`).

DYNAMODB

- **Create a Table**:
  - Search for **DynamoDB**.
  - Click **Create table**.

    - **Name**: "ICT".
    - **Partition key**: "JerseyNO" (Number value).
    - Leave other settings default, click **Create table**.
- **Create Items**:
    - Go to **Explore items**, select the table.
    - **Create an item** in the Form section by entering attributes and values.
    - Can also **create items using JSON format**.
- **Perform Operations**:
    - Use **SCAN** to scan items based on filters.
    - Perform **Query operation** based on the Partition key.
    - Go to **PartiQL** to perform **Query in SQL form**.
    - **Select operation** is performed.
    - **Delete operation** is also performed.

IAM Role (for EC2/S3 Permissions)

- **Create IAM Role**:
    - Go to **IAM service** -> **Roles** -> **Create role**.
    - Provide **EC2 Access** for the role.
    - Provide **EC2 Full Access**.
    - Provide **S3 Full Access**.
    - Give a **name** to your role.
- **Associate Role with EC2 Instance**:
    - **Create an EC2 instance** with basic configurations.
    - After instance is created, select it, click **Security**, and **Modify IAM role**.
    - **Select the created IAM role** and click **Update IAM role**.
- **Test Permissions**:
    - Directly connect to the instance using EC2 Instance Connect.
    - Try **creating an S3 bucket** (e.g., "529-stockholm-bucket-001"); it should succeed due to S3FullAccessPermission.
    - Try **creating a DynamoDB table**; it will be access denied since that permission wasn't added to the role.

Creating IAM User

- **Create IAM User (Individual)**:
    - Login to AWS console, go to **IAM** -> **Users** -> **Add user**.
    - Give user name (e.g., "IAM for ec2").
    - Use **custom password**.
    - Click **Attach Policies directly**, search for **AmazonEC2FullAccess** and select it.
    - Click next and **create user**.
- **Create Another User (S3)**:
    - Create another user with name "frontend".
    - Search for **AmazonS3FullAccess** and create the user.
    - **Download the .csv file** (contains credentials).
- **Create User Group**:
    - Go to **User Groups** and **create the group**.
    - Name the group accordingly.
- **Create User and Add to Group**:
    - Go to Users and **create user with name "XYZ"**.
    - **Add user to the group** created before.
- **Test User Permissions**:
    - Open the downloaded CSV file, copy account ID and username.
    - **Sign in with the new user's credentials**.

- For "IAM for ec2" user, try to create an S3 bucket (will get error as it only has EC2 access).
- For "frontend" user, try to create an EC2 instance (will get error as it only has S3 access).
- For "XYZ" user (group-based permissions), try to create S3 (will get error if group doesn't have S3).

Amazon Lex Chatbot

- **Create Bot**:
  - Go to **Amazon Lex** -> **Create bot**.
  - Select **Traditional** creation method.
  - Give bot name.
  - **IAM permissions**: Create new role.
  - Set Coppa to Yes, idle session timeout to default (5min).
  - Add language or leave default.
- **Add Intent**:
  - Go to the bot -> **Intent -> Add intent**.
  - Add empty intent, **intent name: BookHotel**.
  - Add **sample utterances**.
- **Define Slots**:
  - In Slots -> **Add slot** -> **Name, Slot Type, Prompt**.
  - Add slots for City, When, Days of stay.
- **Configure Responses**:
  - Provide an **Initial response** for the chatbot.
  - Add a **Confirmation message**.
  - Make the **Fulfillment message**.
  - Add the **Closing statement**.
- **Add Custom Slot Type (Optional)**:
  - Bot -> **Slot type -> Add slot type**.
  - Add blank slot, give name.
  - **Slot value resolution - expand -> Add slot type values -> Save**.
  - Go to the intent -> Add slot -> Add the custom slot type.
- **Add Card with Button Options**:
  - In Slot -> Advanced options -> Slot prompts -> More slot prompt options -> Add -> **Card group**.
  - Change Fulfillment accordingly.
- **Integrate Images (Optional)**:
  - Go to S3 -> **Create bucket**, add an image, **give all permissions**.
  - Click on slot -> Advanced -> Slot prompts -> More prompt options -> Add -> Add card group.
  - Add the **image URL from bucket** -> Title.
  - Add button -> Update prompt -> Update slot.
- **Add Another Intent and Build/Test**:
  - Add another intent similarly (utterances, slots, closing, fulfillment).
  - **Build -> Test**.
- **Conditional Branching (Optional)**:
  - Go to a slot -> Advanced -> **Conditional branching**.
  - Give a condition, Update the Slot, and **Build and test**.

Amazon Lex with Twilio Integration

- **Create Lex Bot**:
  - Go to **Amazon Lex**, **create bot**, traditional method, name "HotelBooking".
  - Create a **basic role with Amazon Lex permissions**.
- **Add Intent and Slots**:
  - **Add an intent**, give name as "booking".

    - Add **relevant utterances**.
    - **Create slots**: Name "RoomType", add values (single, double, suite), also slots for City, Date.
    - For RoomType, **give card buttons**.
    - Give **confirmation messages**.
- **Prepare Images for Cards**:
    - **Create an S3 bucket** (general purpose).
    - **Upload an object**, edit ACL permissions, and **copy the URL**.
    - **Paste the URL** in the Lex slot card configuration.
- **Twilio Setup**:
    - **Sign into Twilio**, click on Messages, Try WhatsApp.
    - Verify your number is connected.
- **Integrate Lex with Twilio**:
    - Go to Lex and **add a channel**.
    - Select **Twilio SMS** with basic IAM role.
    - Give name and alias name.
    - **Copy and paste Account SID and Authentication Token from Twilio**.
    - After creation, **copy the callback URL**.
    - **Paste it in Twilio Sandbox settings -> When a message comes in**.
- **Test with WhatsApp**:
    - Go to WhatsApp and **copy and paste "forth-southern"** (or initial utterance) to interact with the bot.