Tuesday, November 4, 2025 10:57 AM

@id -> marks id as private key
@generatedValue(strategy=GenerationType.Identity) -> The database will autogenerate the id

Configration management tools are idempotent

Server templating tool ->Docker -> create custom image of a virtual machine or a container. These images contain all the required software and dependencies intalled onem, eliminates the need for installing software -> custom AMIs in AWS, Docker images in dockerhub -> immutable infrastructure unlike configuration management tools.

Terraform -> PROVISIONING infrasturcture -> build manage and destroy configuration-> deploy infrastructures across multiple platforms like pvt or pub cloud (AWS, GCP, Azure)

How does Terraform manage infra across so many platforms? -> Providers -> make manage 3rd party platforms through their API

Terraform uses HCL (hasicorp confg language)
Can be maintained in a version control
Code is declarative -> terraform will take care of going from current state to desired state without worries how to get there.

Init -> identifies providers
Plan -> plans to get to desired state

Resource -> every object terraform manages

Block:

local.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "We love pets!"
}
```

Block ->defined within curly braces ->configuration data in key value format
/root -> new directory -> we will create HCL configuration files - local.tf -> inside this define resource block.

Here, local_file is the resource type. Resource type provides 2 bits info ->local - provider, file - type of resource.

"pet" is the reosurce name.

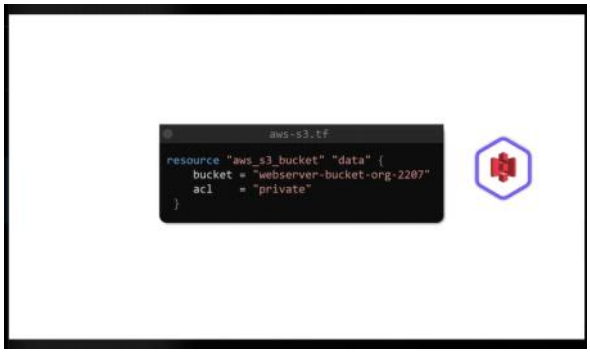
Within braces, we write arguments for resource.

1st argument - filename -> absoulte path.
2nd argument - content -> value.

Now, we have complete HCL configuration file by the name pet.txt
Terraform init will install a plugin called local- so we are ready to create the resource.

aws-ec2.tf

```
resource "aws_instance" "webserver" {
    ami = "ami-0c2f25c1f66a1ff4d"
    instance_type = "t2.micro"
}
```



What Are File Permissions?

In Linux/Unix systems, **file permissions** control **who can read, write, or execute** a file or directory.
There are **three types of permissions**:

- **r** = read (view the contents)
- **w** = write (modify the contents)
- **x** = execute (run the file as a program)

And **three types of users**:

1. **Owner** – the user who owns the file
2. **Group** – users in the same group as the file
3. **Others** – everyone else

Numeric (Octal) Permission Codes

Each permission is represented by a number:

Permission	Binary	Decimal
r	100	4
w	010	2
x	001	1
-	000	0

So, you **add** the values to get the permission for each user type.

Example: chmod 754 file.txt

Breakdown:

- **7** = 4 (read) + 2 (write) + 1 (execute) → rwx (owner)
- **5** = 4 (read) + 0 (no write) + 1 (execute) → r-x (group)
- **4** = 4 (read) + 0 + 0 → r-- (others)

So the permission string is: rwxr-xr--

Symbolic Representation

You might also see permissions like this when you run ls -l:
-rwxr-xr-- 1 user group file.txt

- The first character (-) indicates it's a file (could be d for directory).
- The next 9 characters are grouped as:
  - rwx → owner
  - r-x → group
  - r-- → others

Common Permission Codes

Code	Meaning	Symbolic
777	Full access to all	rwxrwxrwx
755	Owner: all, others: read + execute	rwxr-xr-x
644	Owner: read + write, others: read	rw-r--r--
700	Owner only access	rwx------

Ah, I see where the confusion is coming from — you're referring to **four-digit file permission codes** like "0700" or "0755". Let's clarify what the **first digit** means in a **four-digit permission code**.

🔢 Four-Digit File Permission Code: 0###

- The **last three digits** (###) are the standard permissions for:
  - **Owner**
  - **Group**
  - **Others**
- The **first digit** (the leading 0 in 0700) is **not a user** — it represents **special permission bits**.

🌓 Special Permission Bits (First Digit)

Value	Meaning
0	No special permissions
1	**Sticky bit**
2	**Setgid** (Set group ID)
4	**Setuid** (Set user ID)

These can be **combined** by adding the values:

- 7 = 4 + 2 + 1 → Setuid + Setgid + Sticky bit

🔍 Example: 4755

- 4 → Setuid is set
- 7 → Owner: rwx
- 5 → Group: r-x
- 5 → Others: r-x

This means:

- The file runs with the **owner's privileges** (Setuid).
- Normal permissions apply to group and others.

🔧 Common Use Case

- /usr/bin/passwd has permission -rwsr-xr-x (or 4755)
  - The s in rws means **Setuid** is active.
  - This allows users to change their password (which modifies /etc/shadow) even though they don't have write access to it — because the binary runs with root privileges.

So to answer your question directly:

Who is the "other user" in a 4-digit permission like 0700?

There is **no fourth user**. The **first digit** is for **special permissions**, not a user category.
Would you like a visual breakdown or a cheat sheet for these special bits?

From <https://www.microsoft365.com/chat?version=19.2511.33071.0&capabilities=interopPromise,suspendOnClose,autoStart&client-request-id=c1e9d5db-187c-6b44-bfac-7f41f19acd51>

Now, local.tf ->It is a configuration file.
Confg. dir. Can contain more than one cfg file.

Terraform's plugin-based architecture supports hundreds of infrastructure platforms, and the providers are distributed by HashiCorp via the Terraform Registry.

Parameterizing with Variable instead of hardcoding values

Variables.tf

```
variable "filename" {
  default = "/root/pets.txt"
}

variable "content" {
  default = "We love pets!"
}

variable "prefix" {
  default = "Mrs"
}

variable "separator" {
  default = "."
}

variable "length" {
  default = "1"
}
```

Main.tf:

```
# main.tf
resource "local_file" "pet" {
  filename = var.filename
  content  = var.content
}

resource "random_pet" "my-pet" {
  prefix    = var.prefix
  separator = var.separator
  length    = var.length
}

# variables.tf
variable "filename" {
  default = "/root/pets.txt"
}

variable "content" {
  default = "We love pets!"
}
```

Creating an EC2 Instance with Variables

Here's an additional example of using input variables to create a
configuration demonstrates the same variable usage pattern, ev
unfamiliar:

```
resource "aws_instance" "webserver" {
  ami           = var.ami
  instance_type = var.instance_type
}

variable "ami" {
  default = "ami-0edab43b6fa892279"
}

variable "instance_type" {
  default = "t2.micro"
}
```

Variable Blocks

Apart from default, variable block can also contian several parameters.

```
variable "filename" {
  default     = "/root/pets.txt"
  type        = string
  description = "The path of the local file"
}

variable "content" {
  default     = "I love pets!"
  type        = string
  description = "The content of the file"
}

variable "prefix" {
  default     = "Mrs"
  type        = string
  description = "The prefix to be set"
}

variable "separator" {
  default = "."
}
```

List Variables

A list is an ordered collection of values where ea
consider a variable that uses a list of prefixes:

```
variable "prefix" {
  default = ["Mr", "Mrs", "Sir"]
  type    = list(string)
}

resource "random_pet" "my-pet" {
  prefix = var.prefix[0]
}
```

In this configuration:

- `var.prefix[0]` returns "Mr"
- `var.prefix[1]` returns "Mrs"
- `var.prefix[2]` returns "Sir"

Map Variables

Maps allow you to define key-value pairs for storing related data. For

```
variable "file_content" {
  type    = map(string)
  default = {
    "statement1" = "We love pets!"
    "statement2" = "We love animals!"
  }
}
```

To reference a specific value from this map in a resource, use the key

```
resource "local_file" "my-pet" {
  filename = "/root/pets.txt"
  content  = var.file_content["statement2"]
}
```

List and Map Type Constraints

You can enforce type constraints on lists and maps to

```
variable "prefix" {
  default = ["Mr", "Mrs", "Sir"]
  type    = list(string)
}

variable "numbers" {
  default = [1, 2, 3]
  type    = list(number)
}
```

```
variable "cats" {
  default = {
    "color" = "brown"
    "name"  = "bella"
  }
  type = map(string)
}

variable "pet_count" {
  default = {
    "dogs"     = 3
    "cats"     = 1
    "goldfish" = 2
  }
  type = map(number)
}
```

Using variables in terraform

If a variable does not have a default value or if you want to override an existing default, Terraform will prompt you for a value during terraform apply.

Using Variable definition files

☑ Why Use terraform.tfvars?

You **don't have to** use terraform.tfvars — but it's **very useful** when:

1. You want to override default values

Instead of editing variables.tf, you can just assign values in terraform.tfvars.

2. You want to separate logic from data

Keep your infrastructure logic (main.tf) clean and put actual values in a separate file.

3. You want to switch environments easily

You can create:

- dev.tfvars
- prod.tfvars
- test.tfvars

Then run:

terraform apply -var-file="prod.tfvars"

This makes your setup **modular and scalable**.

🧨 Example Without terraform.tfvars

You can do everything in variables.tf like this:

Terraform
variable "filename" {
 default = "games.txt"
}

variable "content" {
 default = "FIFA 21"
}
Show more lines
And use var.filename and var.content in main.tf.
☑ This works perfectly for **simple or static setups**.

🧠 So When Should You Use terraform.tfvars?

Use it when:

- You want to **change values without editing code**
- You have **multiple environments**
- You want to **keep secrets or sensitive values** out of version-controlled files (you can even use .tfvars files in .gitignore)

When managing many variables, it becomes practical to store their values in a dedicated variable definition file. These files typically have a .tfvars or .tfvars.json extension.

Using Variable Definition Files

When managing many variables, it becomes practical to store their values in a dedicated variable definition file. These files typically have a `.tfvars` or `.tfvars.json` extension. For example, you can create a file named `terraform.tfvars` with the following contents:

```
filename = "/root/pets.txt"
content  = "We love pets!"
prefix   = "Mrs"
separator = "."
length   = "2"
```

Terraform automatically loads files named `terraform.tfvars`, `terraform.tfvars.json`, or files with extensions like `.auto.tfvars` or `.auto.tfvars.json`. If you use a differently named file (e.g., `variables.tfvars`), be sure to specify it explicitly with the `-var-file` flag:

```
$ terraform apply -var-file="variables.tfvars"
```

This approach centralizes your variable definitions and simplifies the management of Terraform environments.

Resource Attributes

Consider a scenario where you want to use the output of one resource as an input for another.

1. Create a file resource

```terraform
resource "local_file" "games" {
  filename = "${path.module}/favorite-games.txt"
  content  = "FIFA 21"
}
```

2. Use its attribute in another resource

```terraform
resource "null_resource" "print_file_path" {
  provisioner "local-exec" {
    command = "echo File created at ${local_file.games.filename}"
  }
}
```

Here:

- local_file.games.filename is an **attribute** of the local_file resource.
- It's used as input in the null_resource to print the file path.

when you pass the output of one resource (like a random pet) to another resource (such as a local file), Terraform understands that the random pet must be created before the local file -> implicit dependency

Below is an example of an implicit dependency:

```
resource "local_file" "pet" {
  filename = var.filename
  content  = "My favorite pet is ${random_pet.my-pet.id}"
}

resource "random_pet" "my-pet" {
  prefix    = var.prefix
  separator = var.separator
  length    = var.length
}
```

Output Variables

Terraform output variables are a powerful feature that allow you to store the results of expressions from your configuration files for later use.

Below is an example configuration:

```
resource "local_file" "pet" {
  filename = var.filename
  content  = "My favorite pet is ${random_pet.my-pet.id}"
}

resource "random_pet" "my-pet" {
  prefix    = var.prefix
  separator = var.separator
  length    = var.length
}

output "pet-name" {
  value = random_pet.my-pet.id
  desc  = ""
}
```

Terraform State

Terraform maintains a state file, which is how it tracks that the resource is already provisioned.

After the initial successful terraform apply, an additional file named terraform.tfstate is created in the project directory.

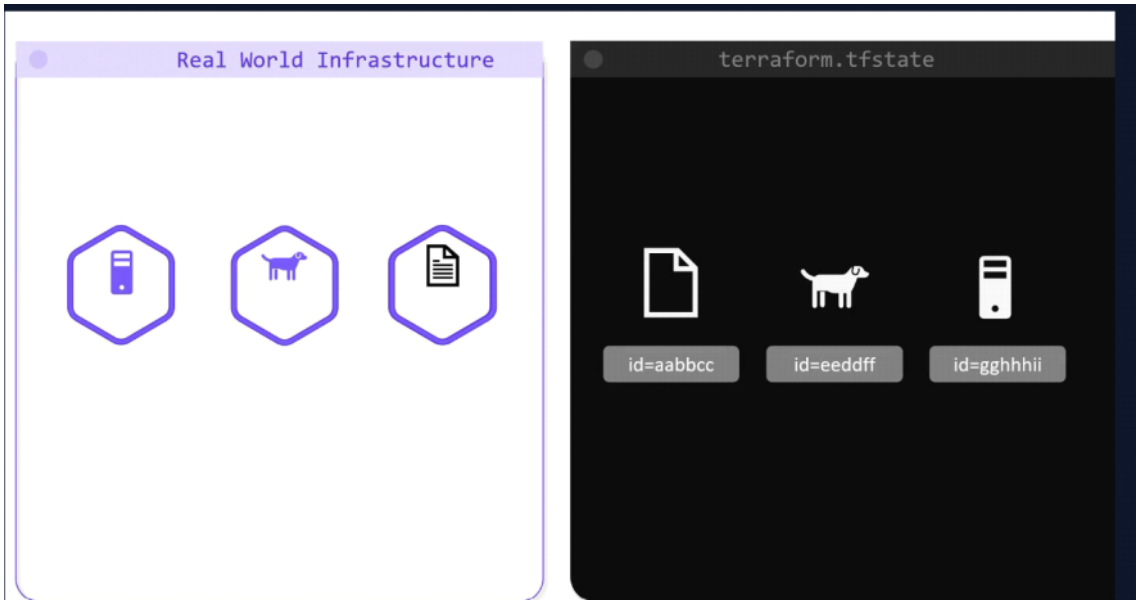
Inspecting `terraform.tfstate` reveals a detailed record of the infrastructure, including resource IDs, provider information, and resource attributes:

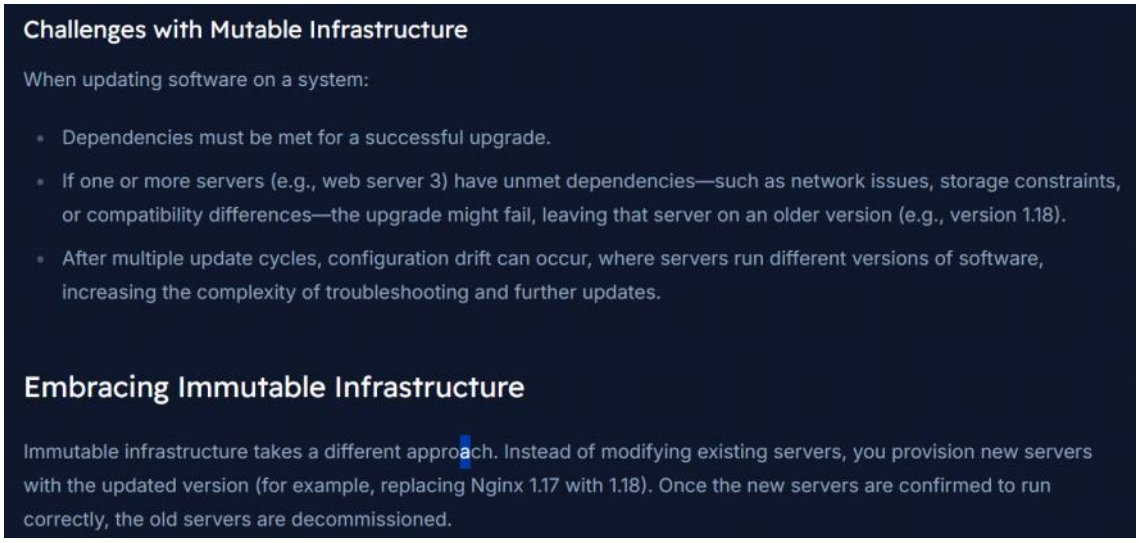

```
{
  "version": 4,
  "terraform_version": "0.13.0",
  "serial": 1,
  "lineage": "e35dde72-a943-de50-3c8b-1df8986e5a31",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "local_file",
      "name": "pet",
      "provider": "provider[\"registry.terraform.io/hashicorp/local\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "I love pets!",
            "content_base64": null,
            "directory_permission": "0777",
            "file_permission": "0777",
            "filename": "/root/pets.txt",
            "id": "7e4db4fbfdcb108dd04692062bae39bd1e1b68",
            "sensitive_content": null
          },
          "private": "bnVzBA=="
        }
```


Now, if we update configurations in variables.tf,
running terraform apply causes Terraform to refresh the state and detect a difference between the new configuration and the existing state. Consequently, Terraform decides the resource must be replaced

After applying these changes, Terraform deletes the old resource and creates a new one with a different unique ID. The updated terraform.tfstate now reflects the new state

Now that the configuration file and the state file are in sync, any subsequent runs of terraform apply will report that no changes are necessary.





Benefits of Immutable Infrastructure

- Easier to version infrastructure and roll back to previous releases.
- Consistent environments reduce the complexity of governance and maintenance.
- Reliable deployment pipelines, as each release is a fresh creation rather than an update

By default, Terraform replaces the resource.

By default, when Terraform updates a resource, it treats it as immutable. This means the existing resource is deleted before a new one is created with the updated configuration. For example, if you update the file permissions on a local file resource from 0777 to 0700 and then run terraform apply, Terraform will first delete the old file and then create a new one.

The create_before_destroy Rule

The `create_before_destroy` lifecycle rule instructs Terraform to create a new resource before deleting the old one. This is particularly useful when maintaining service availability is critical.

```
resource "local_file" "pet" {
  filename        = "/root/pets.txt"
  content         = "We love pets!"
  file_permission = "0700"

  lifecycle {
    create_before_destroy = true
  }
}
```

The prevent_destroy Rule

In some cases, you might want to ensure a resource is never accidentally deleted—even if a configuration change would normally force a replacement. Terraform allows you to achieve this using the prevent_destroy rule.

- The ignore_changes rule allows you to specify attributes that Terraform should ignore during state comparisons, accommodating external changes.

Datasource

```
$ cat /root/dog.txt
Dogs are awesome!
```

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content  = data.local_file.dog.content
}

data "local_file" "dog" {
  filename = "/root/dog.txt"
}
```

Resource	Data Source
Keyword: **resource**	Keyword: **data**
Creates, Updates, Destroys Infrastructure	Only **Reads** Infrastructure
Also called **Managed Resources**	Also called **Data Resources**

Traditional Loop Approach

In traditional scripting, such as with bash, you might use a loop to create multiple files. The example below creates three empty files (`pet1`, `pet2`, and `pet3`) in the `/root` directory:

```
#!/bin/bash
for i in {1..3}
do
    touch /root/pet${i}
done
```

After running the above script, listing the directory contents may produce:

```
$ ls -ltr /root/
-rw-r--r-- 1 root root 0 Sep  9 02:04 pet2
-rw-r--r-- 1 root root 0 Sep  9 02:04 pet1
-rw-r--r-- 1 root root 0 Sep  9 02:04 pet3
```

Creating Multiple Instances with a Static Count

One of the simplest ways to create multiple instances of a resource is by using a static count. Below is an example that creates three instances of a local file resource:

```
resource "local_file" "pet" {
  filename = var.filename
  count    = 3
}
```

The variable is defined as follows:

```
variable "filename" {
  default = "/root/pets.txt"
}
```

When you run `terraform plan`, Terraform plans to create three resources:

Here,
Terraform creates the same file three times rather than three unique files.

Each resource is indexed as pet[0], pet[1], and pet[2]. Although three resources are created, all instances share the same file name

Creating Unique Resources by Using a List Variable

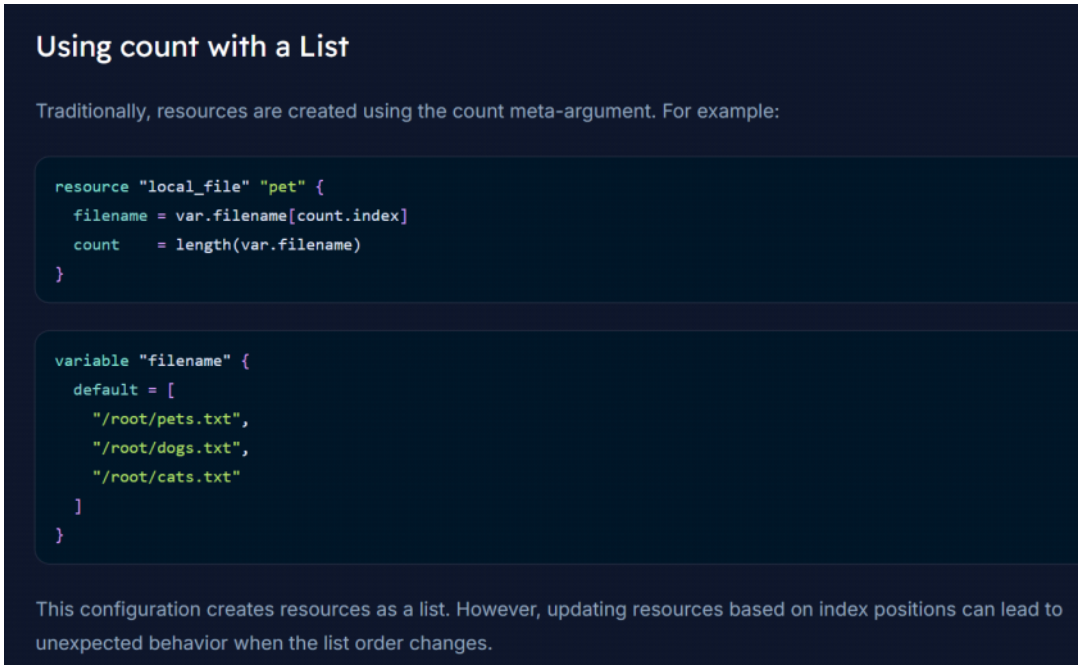
To generate unique resources, update the variable definition to a list and reference each element using `count.index`. Here is the modified configuration:

```
resource "local_file" "pet" {
  filename = var.filename[count.index]
  count    = 3
}
```

Define the list variable as follows:

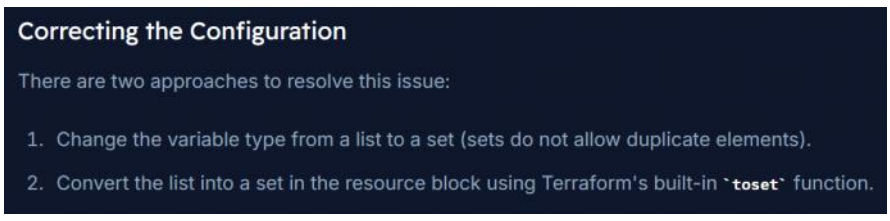
```
variable "filename" {
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

A drawback of the previous approach is its fixed count.



Transitioning to for_each

The for_each argument only supports a map or a set of strings – not a list of strings



Below is an updated configuration that uses the `toset` function:

```
resource "local_file" "pet" {
  filename = each.value
  for_each = toset(var.filename)
}

variable "filename" {
  type    = list(string)
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

When you run terraform plan now, Terraform will indicate that three resources will be created

Updating Resources by Removing an Element

Let's simulate updating the configuration by removing an element. For example, removing `/root/pets.txt` from the list changes the configuration to:

```
resource "local_file" "pet" {
  filename = each.value
  for_each = toset(var.filename)
}
```

```
variable "filename" {
  type    = list(string)
  default = [
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

Running `terraform plan` with this updated variable shows that only the resource associated with `/root/pets.txt` will be destroyed:

Specifying a Specific Provider Version

To enforce a particular version of the local provider (for example, version 1.4.0), include a `terraform` block with a `required_providers` sub-block. This block explicitly instructs Terraform which version to install. Consider the following configuration:

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "1.4.0"
    }
  }
}

resource "local_file" "pet" {
  filename = "/root/pet.txt"
  content  = "We love pets!"
}
```

For developers like Max, Abdul, and Lee who require access to both AWS EC2 and S3 services, a common practice is to create an IAM group (e.g., "Developer Group") and attach policies such as AmazonEC2FullAccess and AmazonS3FullAccess.

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Introduction-to-IAM>

AWS services, such as an EC2 instance, do not inherently possess permissions to interact with other AWS resources (for example, accessing an S3 bucket).

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Introduction-to-IAM>

To enable an AWS service to interact with another resource, you create an IAM role.

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Introduction-to-IAM>

Creating Custom IAM Policies

In addition to AWS managed policies, you can create custom IAM policies to match specific operational requirements. For instance, if you want a user to be able to create and delete tags on an EC2 instance, you could define a custom policy as shown below:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Resource": "*"
        }
    ]
}
```

This custom policy grants the user permission exclusively to perform the "ec2:CreateTags" and "ec2:DeleteTags" actions on any EC2 resources. Once created, this policy can be assigned within your AWS account to enforce specific permissions.

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Introduction-to-IAM>

IAM is a global service—unlike other AWS services, once you create an IAM object (such as a user or group), it is available in every region across your account.

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Demo-IAM>

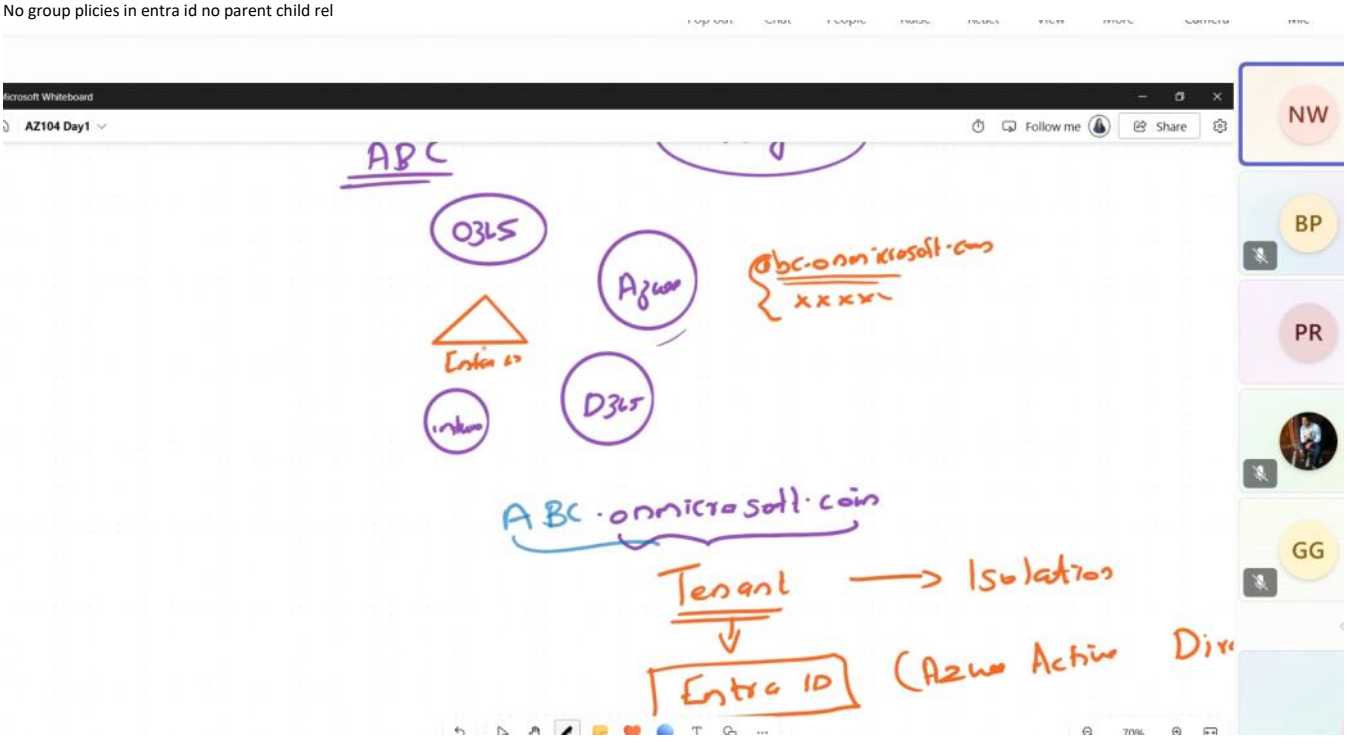
Attaching Policies to a User

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Demo-IAM>

Now, attach an additional policy that grants Lucy full administrative privileges. As the team lead, she needs unrestricted access to all account services. We will attach the AdministratorAccess Policy

From <https://notes.kodekloud.com/docs/Terraform-Basics-Training-Course/Terraform-with-AWS/Demo-IAM>

AZ 104 Day 1

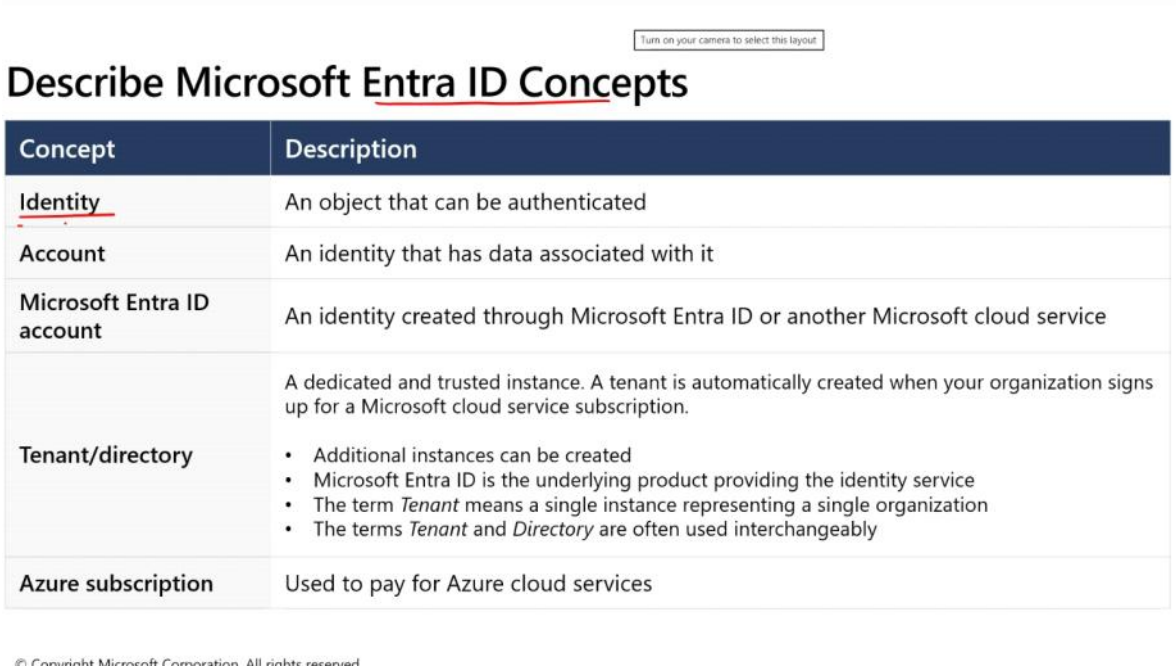
1. Tenant -> renting services from ms. Each tenant is isolation
   ABC.onmicrosoft.com -> entra id (Azure active directory) for authentication.
   Enta id -> authentication and authorization
2. Every tenant has 1 entra id.
3. No group plicies in entra id no parent child rel
4. 

5. IAAS

Authenticatrion protocol ->SAML -> securty assertion markup language
Wsfed
Open id connect

For authorization ->Oauth

On premise Ad



Tenant is automatically created when I sign in any microsoft service in office.

Netflix example -> availing subscriptiion for user-> so creating tenant wont enable us to deploy machines -> we need subscription., to use any azure cloud services.

When you **sign in to a Microsoft service using a work or school account**, especially in an **organization like a company or educational institution**, Microsoft uses something called **Microsoft Entra ID** (formerly Azure Active Directory) to manage identities and access.
Here's what your trainer likely meant:

◇ What is a "Tenant" in Microsoft?

A **tenant** is like a dedicated space or container in Microsoft's cloud for your organization. It holds:

- User accounts
- Security settings
- Licenses
- Access to Microsoft services like Teams, Outlook, SharePoint, etc.

◇ How is it "automatically created"?

- If your organization **already has a Microsoft 365 or Azure subscription**, then the tenant already exists.
- But if someone from your organization **signs up for a Microsoft service for the first time** (like Microsoft 365 or Azure) using a work email, Microsoft may **automatically create a tenant** for that domain (e.g., @yourcompany.com).
- This tenant is then used to manage all users and services under that domain.

◇ Example:

Let's say your company email is arka@companyname.com. If no one from companyname.com has ever signed up for Microsoft services before, when you sign in, Microsoft might create a tenant for companyname.com and associate your account with it.

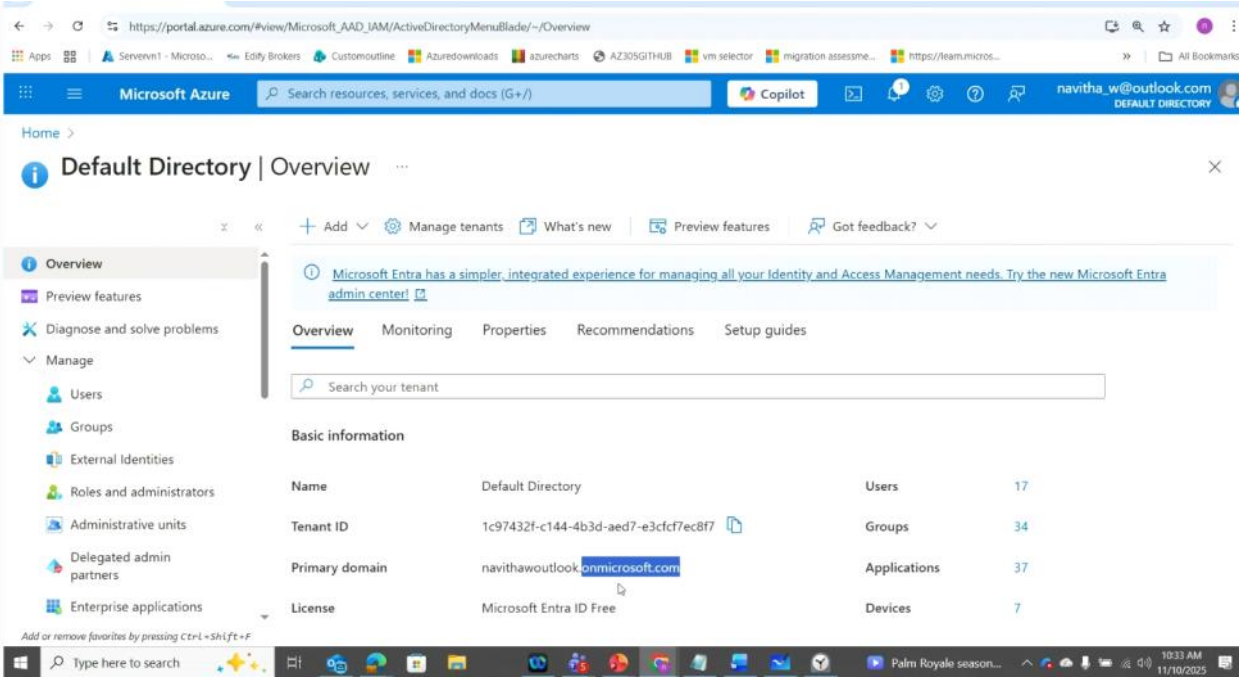
Would you like a simple diagram or visual to help explain this better? Or do you want to know how to check if your organization already has a tenant?

From <https://www.microsoft365.com/chat?version=19.2511.34141.0&capabilities=interopPromise,suspendOnClose,autoStart&client-request-id=914931a4-e3b7-2346-b0b8-8c77b2fe8dce>

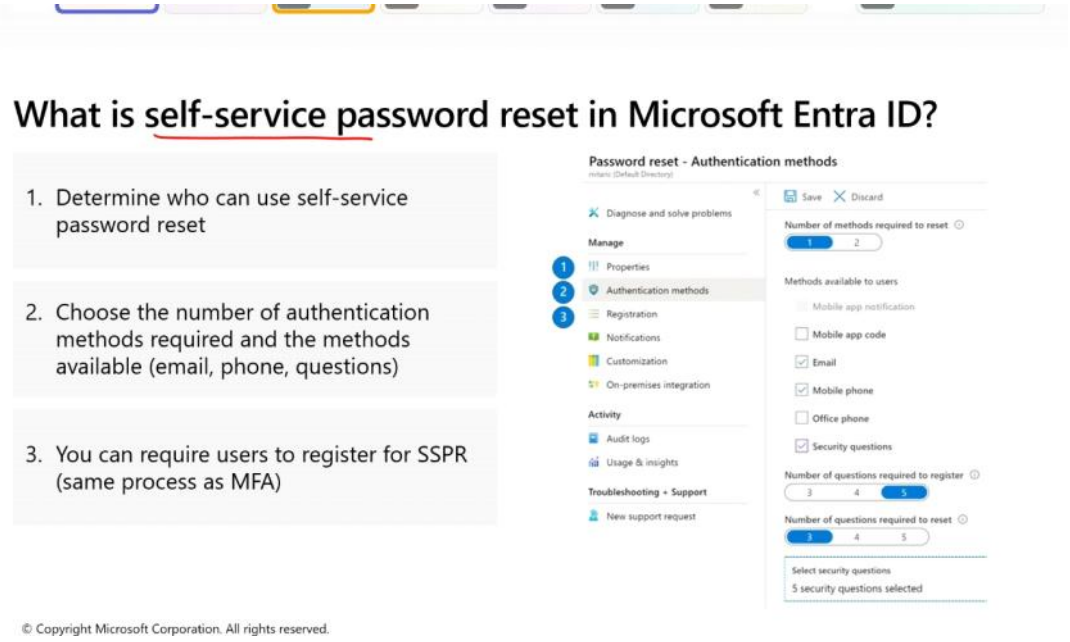
Turn on your camera to select this layout

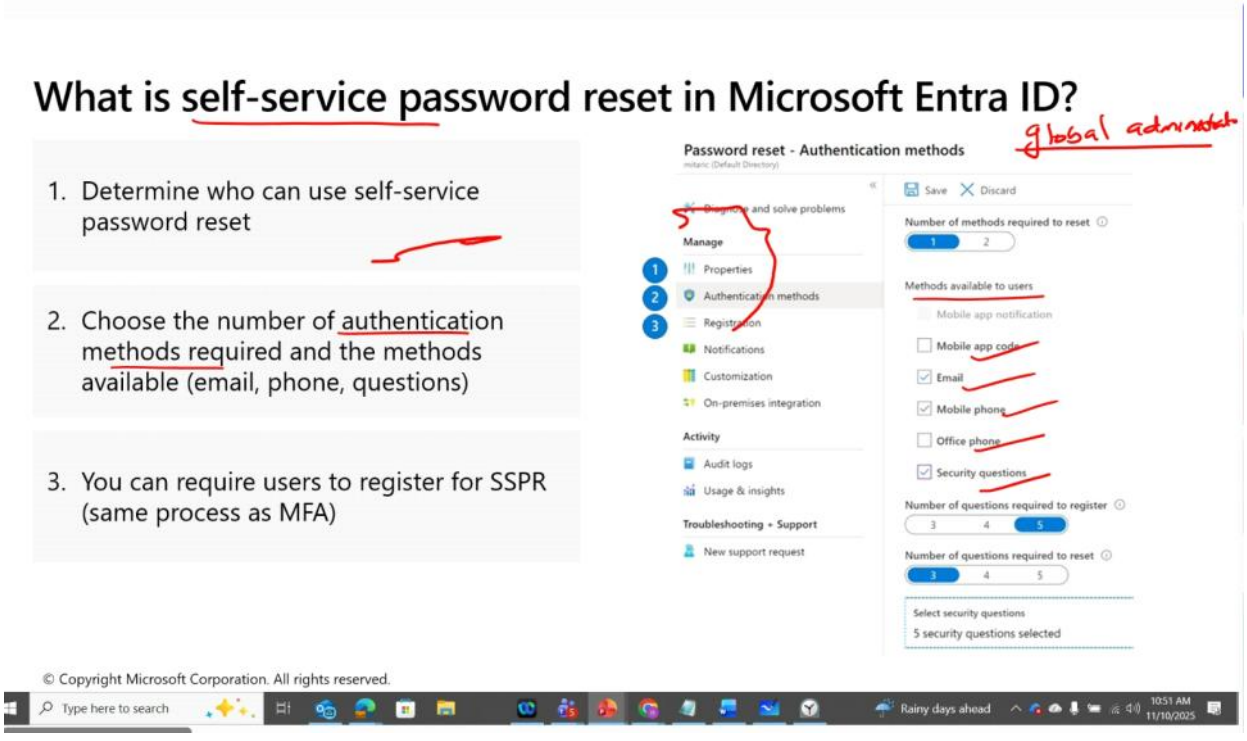
Compare Microsoft Entra ID to Active Directory Domain Services

- Microsoft Entra ID is primarily an identity solution
- Queried using the REST API over HTTP and HTTPS
- Uses HTTP and HTTPS protocols such as SAML, WS-Federation, and OpenID Connect for authentication (and OAuth for authorization)
- Includes federation services, and many third-party services (such as Facebook)
- Microsoft Entra ID users and groups are created in a flat structure, and there are no Organizational Units (OUs) or Group Policy Objects (GPOs)



SSO is a feature of entra id





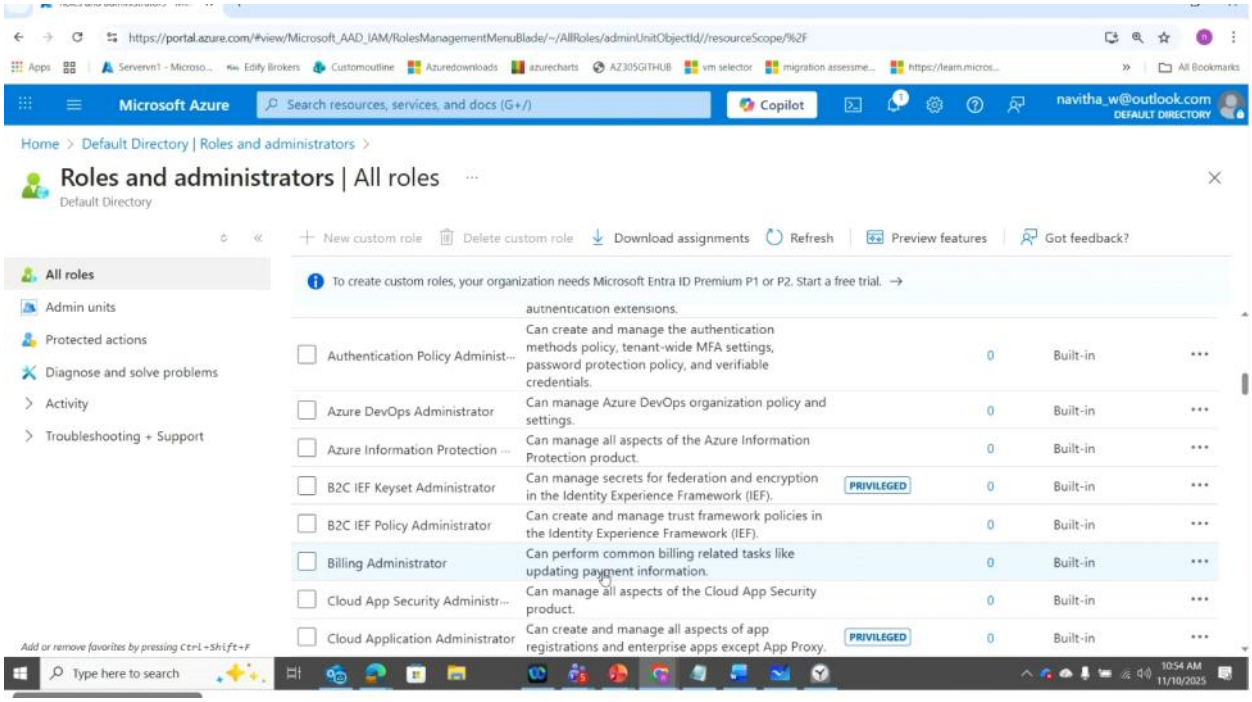
P1 license required for SSPR

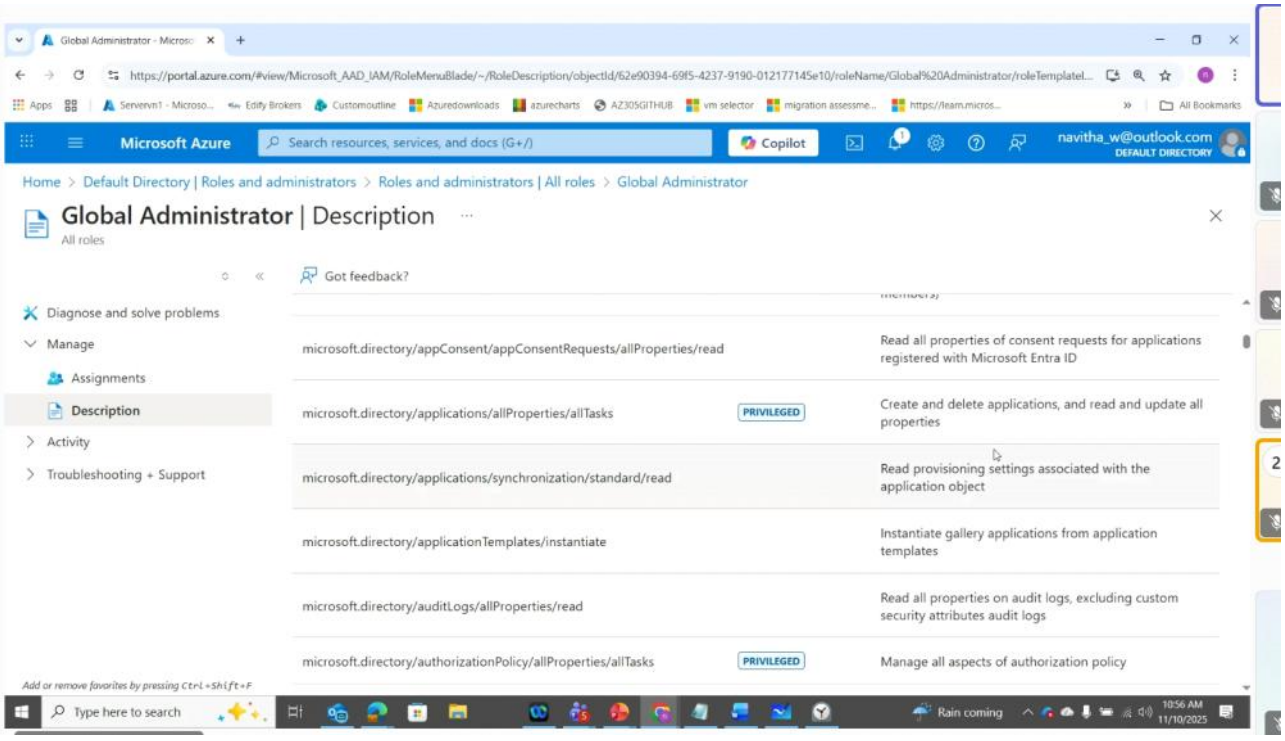
Global administrator -> highest previledge role for entra id

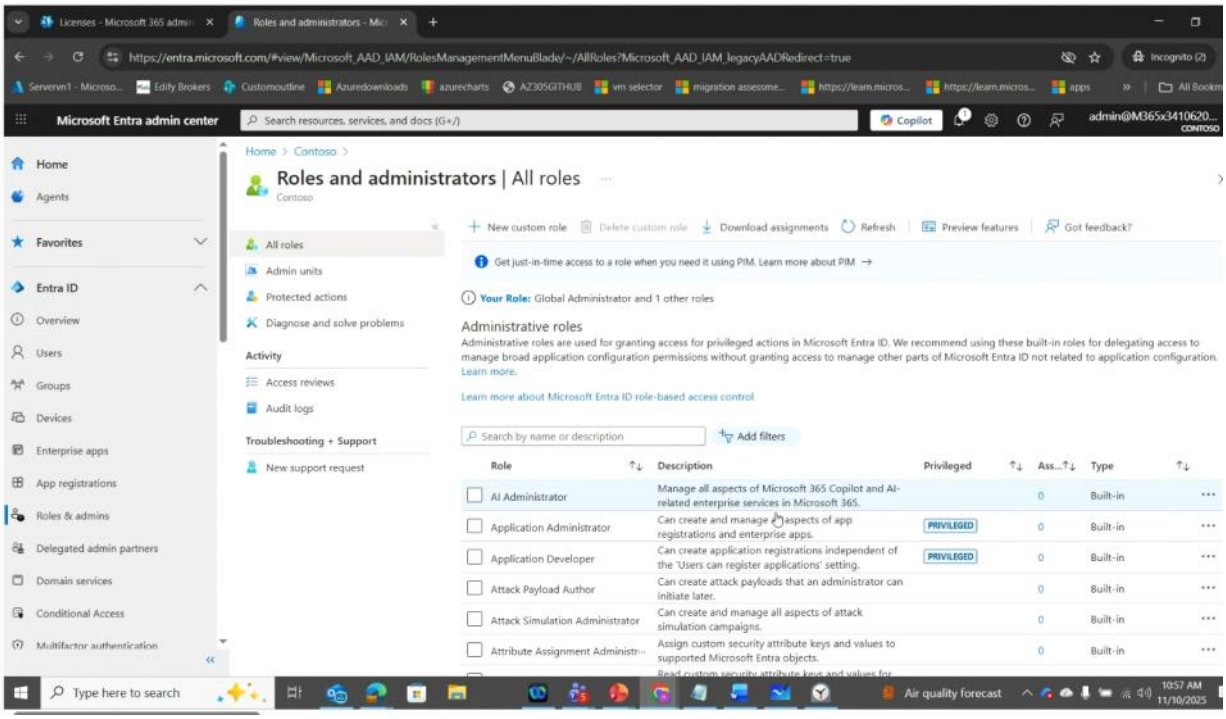
Entra id roles -> for entra id features only

Password admin also is another role.

Go to diff directory to see diff roles ..now we are concerned about only entra id roles


Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
navitha_w@outlook.com
DEFAULT DIRECTORY
Home > Default Directory | Roles and administrators >
Roles and administrators | All roles
Default Directory
New custom role
Delete custom role
Download assignments
Refresh
Preview features
Got feedback?
All roles
Admin units
Protected actions
Diagnose and solve problems
Activity
Troubleshooting + Support
To create custom roles, your organization needs Microsoft Entra ID Premium P1 or P2. Start a free trial.
authentication extensions.
Authentication Policy Administ...
Can create and manage the authentication methods policy, tenant-wide MFA settings, password protection policy, and verifiable credentials.
Azure DevOps Administrator
Can manage Azure DevOps organization policy and settings.
Azure Information Protection ...
Can manage all aspects of the Azure Information Protection product.
B2C IEF Keyset Administrator
Can manage secrets for federation and encryption in the Identity Experience Framework (IEF).
PRIVILEGED
B2C IEF Policy Administrator
Can create and manage trust framework policies in the Identity Experience Framework (IEF).
Billing Administrator
Can perform common billing related tasks like updating payment information.
Cloud App Security Administr...
Can manage all aspects of the Cloud App Security product.
Cloud Application Administrator
Can create and manage all aspects of app registrations and enterprise apps except App Proxy.
Built-in
Add or remove favorites by pressing Ctrl+Shift+F
Type here to search
10:54 AM
11/10/2025


Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
navitha_w@outlook.com
DEFAULT DIRECTORY
Home > Default Directory | Roles and administrators > Roles and administrators | All roles > Global Administrator
Global Administrator | Description
All roles
Got feedback?
Diagnose and solve problems
Manage
Assignments
Description
Activity
Troubleshooting + Support
microsoft.directory/appConsent/appConsentRequests/allProperties/read
Read all properties of consent requests for applications registered with Microsoft Entra ID
microsoft.directory/applications/allProperties/allTasks
PRIVILEGED
Create and delete applications, and read and update all properties
microsoft.directory/applications/synchronization/standard/read
Read provisioning settings associated with the application object
microsoft.directory/applicationTemplates/instantiate
Instantiate gallery applications from application templates
microsoft.directory/auditLogs/allProperties/read
Read all properties on audit logs, excluding custom security attributes audit logs
microsoft.directory/authorizationPolicy/allProperties/allTasks
PRIVILEGED
Manage all aspects of authorization policy
Add or remove favorites by pressing Ctrl+Shift+F
Type here to search
Rain coming
10:56 AM
11/10/2025



Entra id is a service, not a role

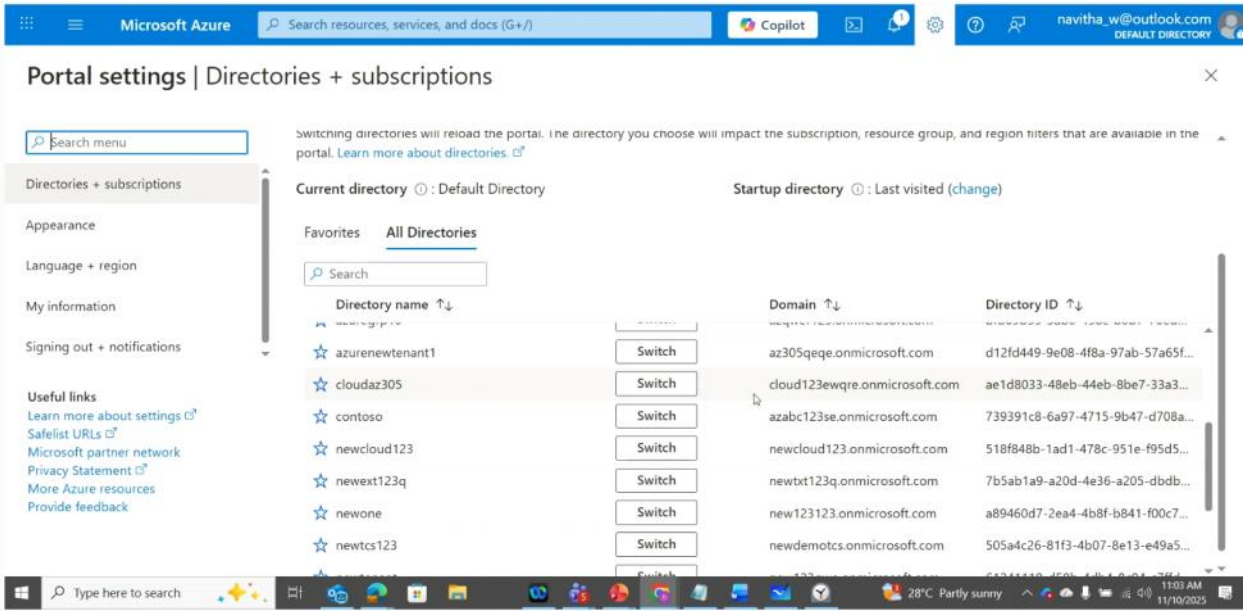
☑ **Entra ID Roles (formerly Azure AD roles)**
These are specific to identity management:

Role Name	What It Can Do
Global Administrator	Full control over Entra ID (can assign roles, reset passwords, etc.).
User Administrator	Can manage users and groups.
Security Administrator	Can manage security-related features.
Application Administrator	Can manage app registrations and permissions.

Not all users in one tenant

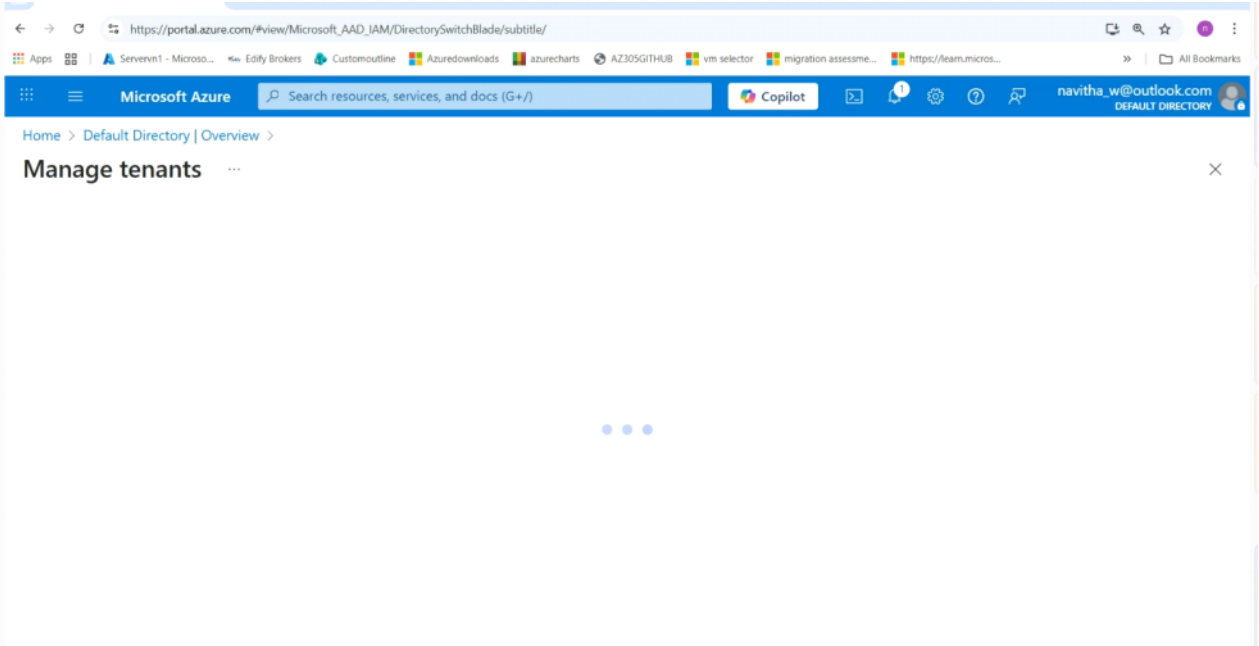
1 tenant -> lti mindtree bangalore employees
Another tenant -> lti kolkata

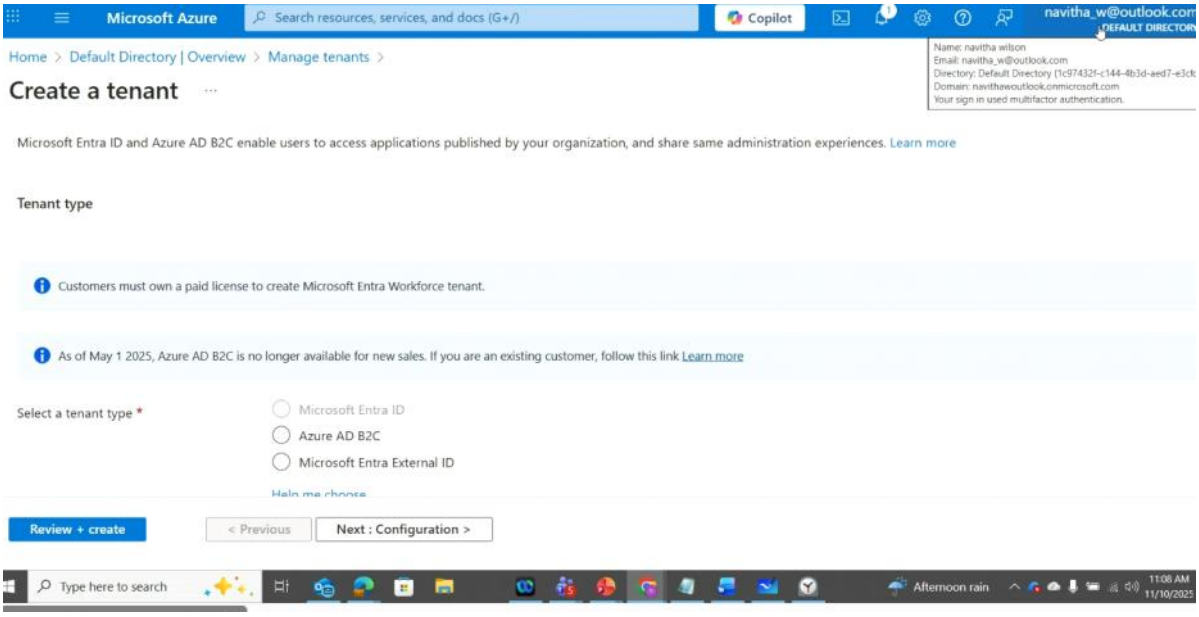
Tenants are given below:



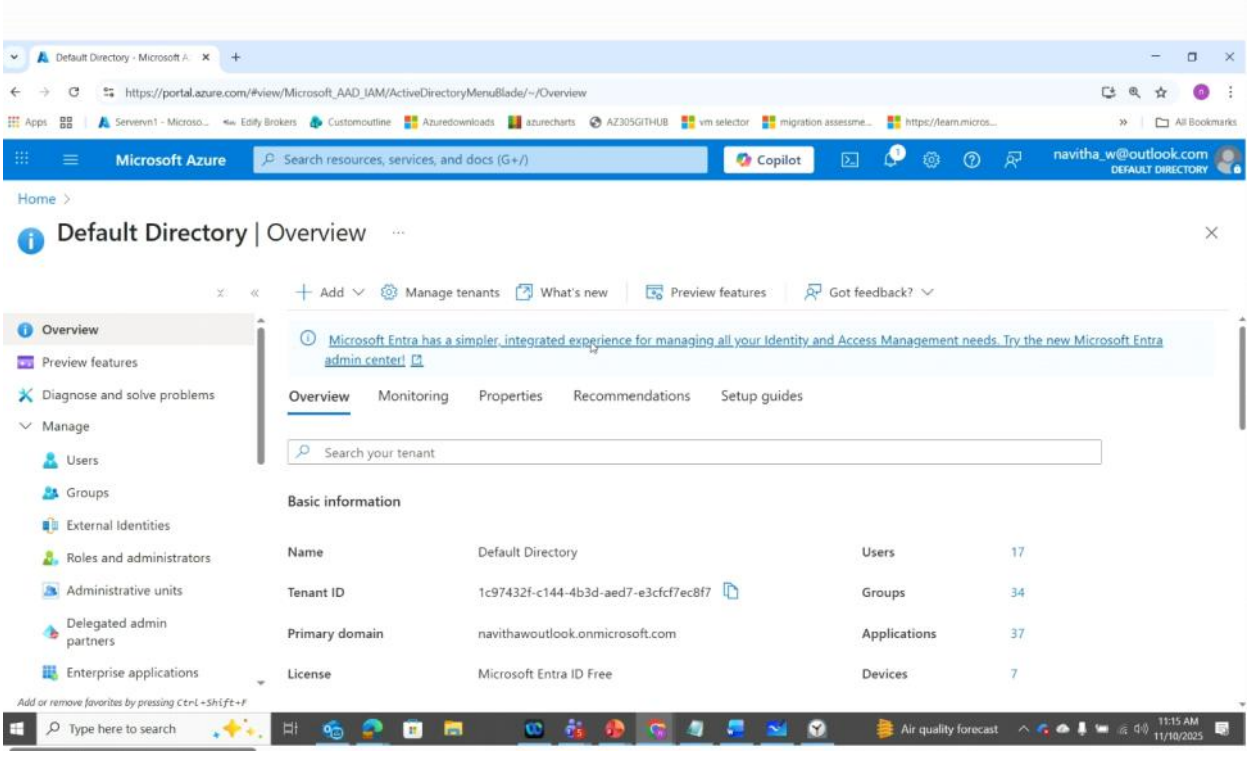
Creating Tenants:

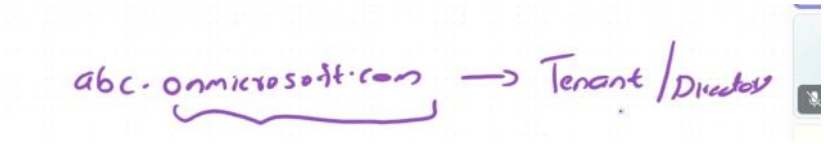
Resource group and tenant diff



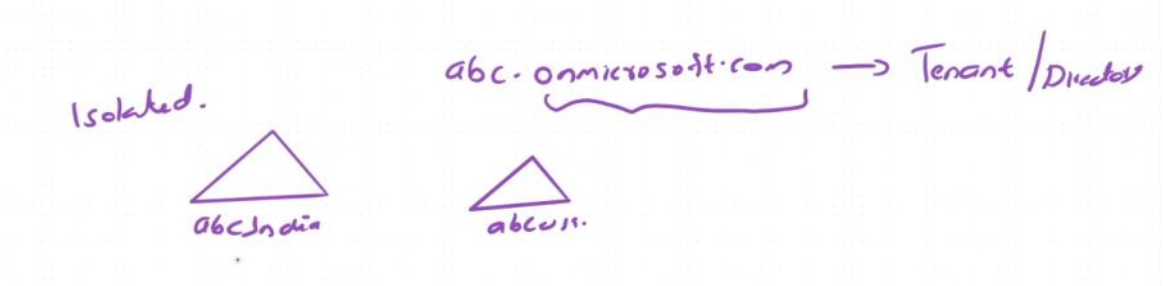


Creating tenants and switching it in azure portal settings
All tenants are isolated

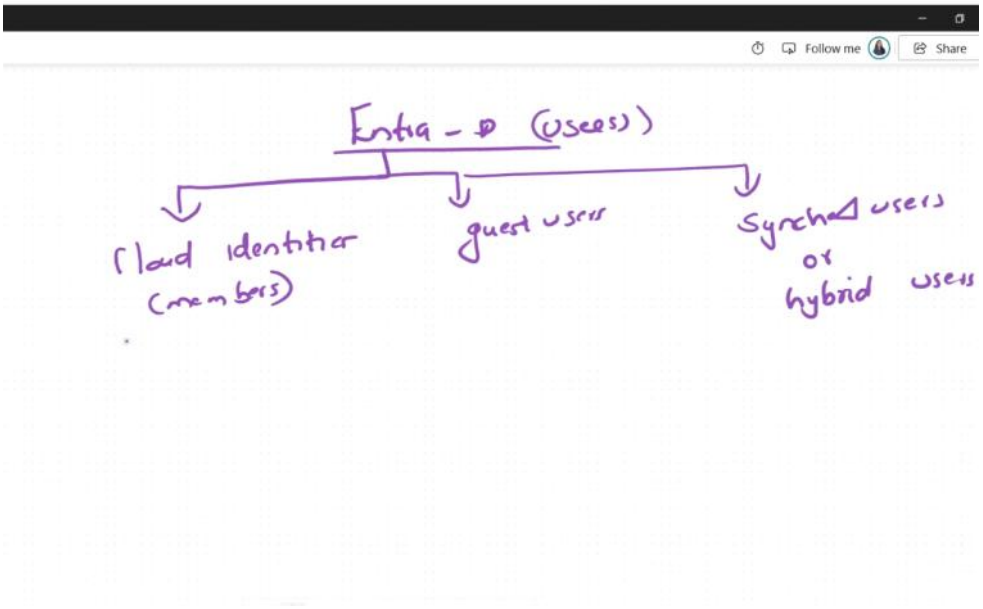




Default tenant

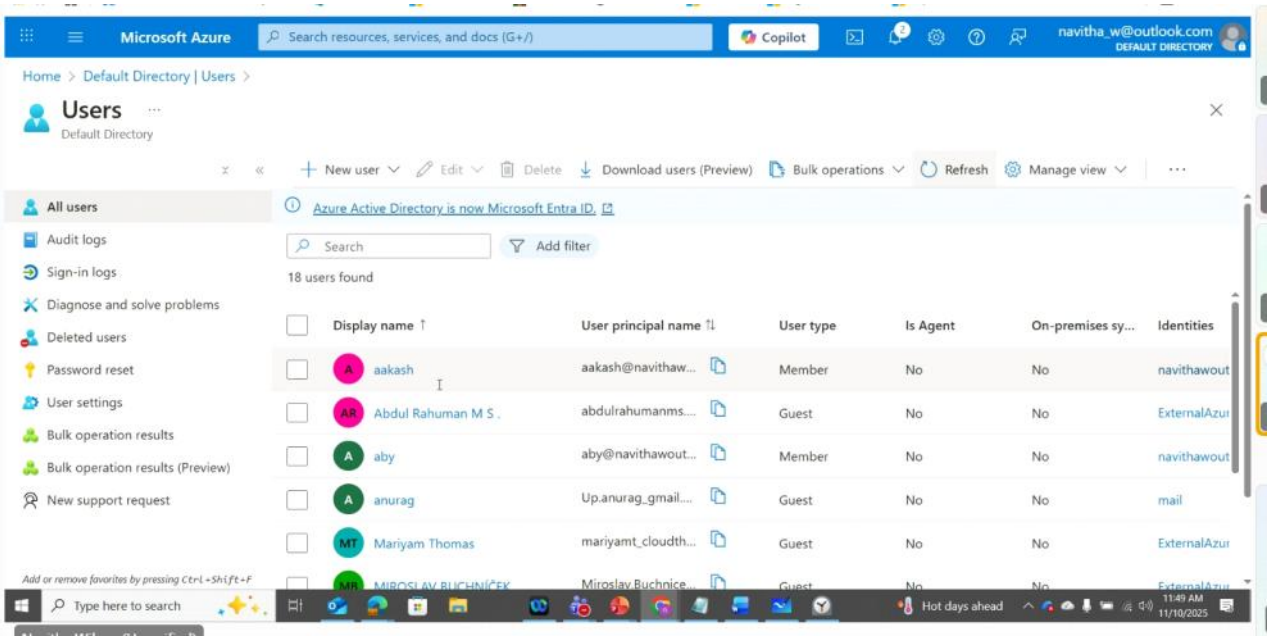


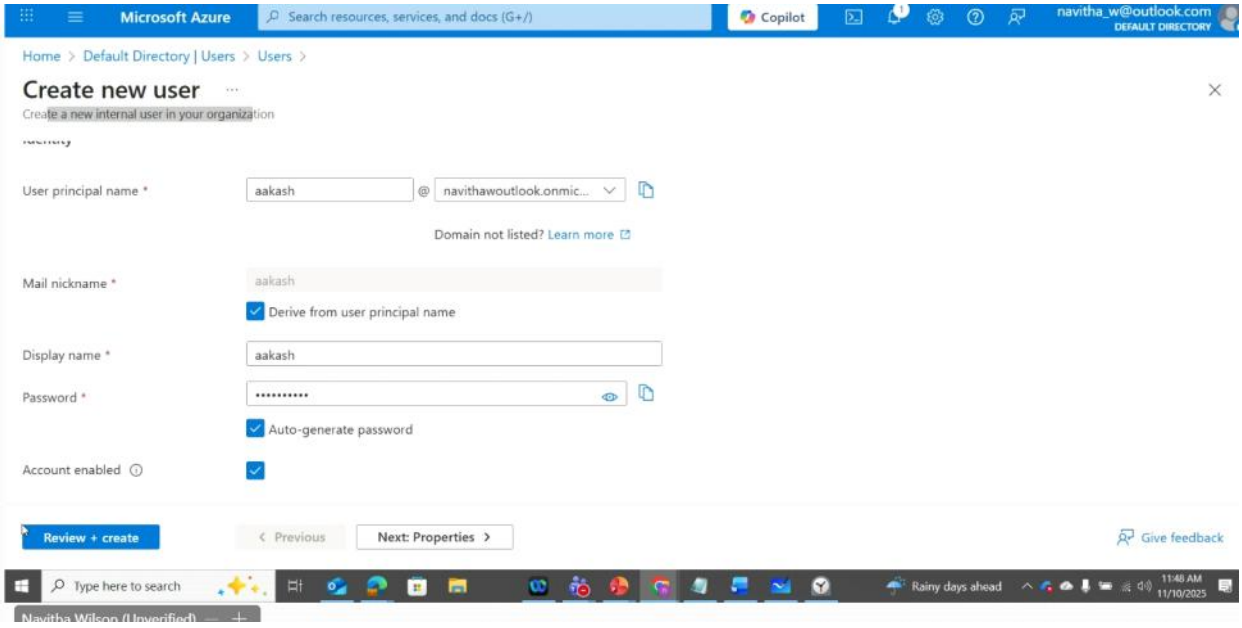
Tenants created under my tenant. They are isolated, and authentication through entra -id.

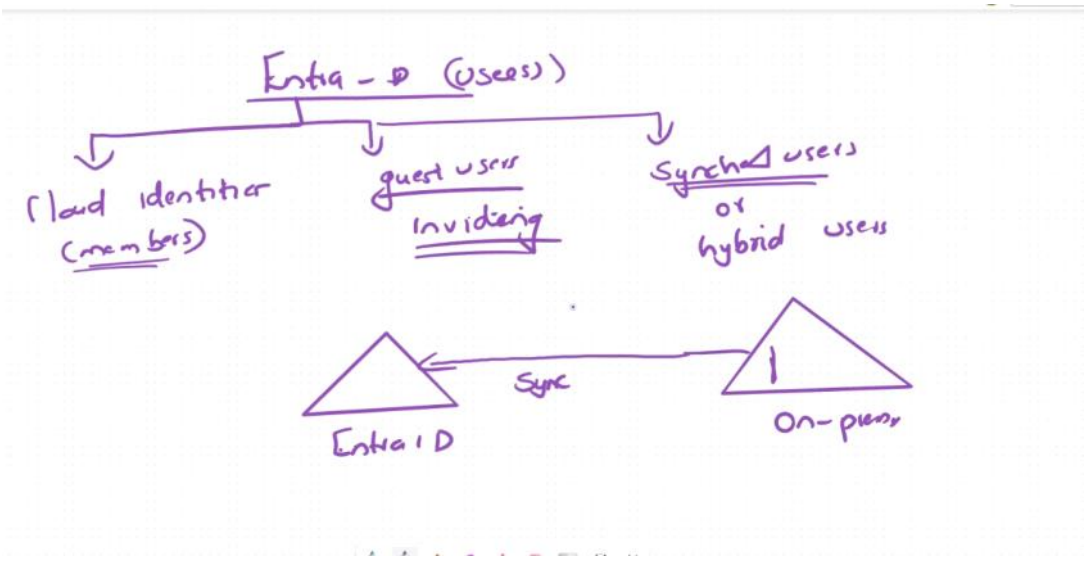


Identities in cloud itself -> cloud identities

Cloud idenity:

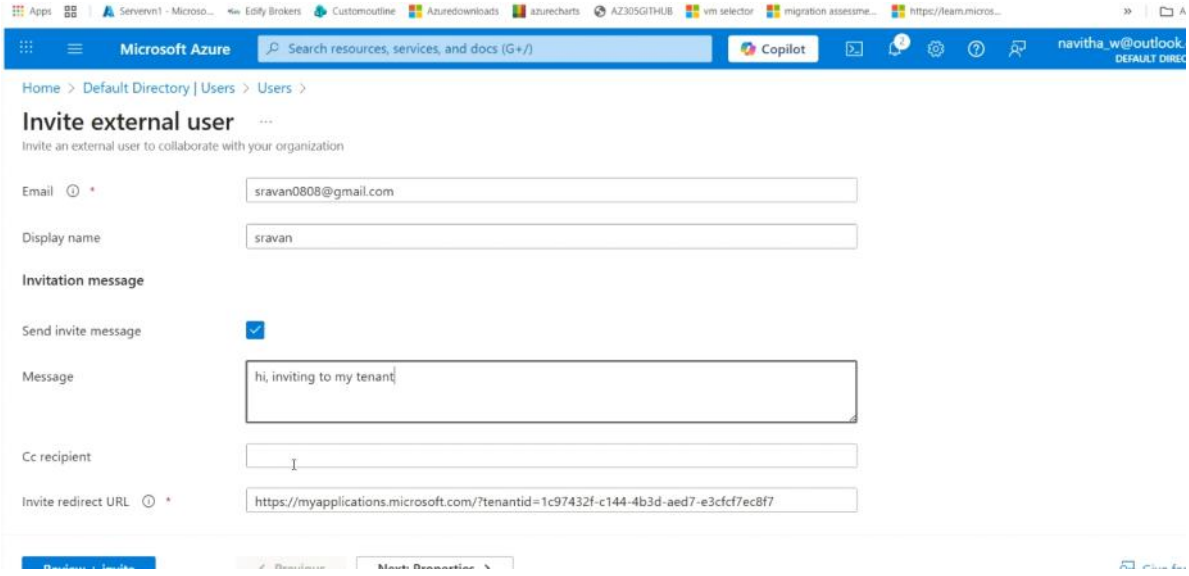

Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
navitha_w@outlook.com
DEFAULT DIRECTORY
Home > Default Directory | Users >
Users
Default Directory
New user
Edit
Delete
Download users (Preview)
Bulk operations
Refresh
Manage view
All users
Audit logs
Sign-in logs
Diagnose and solve problems
Deleted users
Password reset
User settings
Bulk operation results
Bulk operation results (Preview)
New support request
Azure Active Directory is now Microsoft Entra ID.
Search
Add filter
18 users found
Display name
User principal name
User type
Is Agent
On-premises sy...
Identities
aakash
aakash@navithaw...
Member
No
No
navithawout
Abdul Rahuman M S .
abdulrahumanms....
Guest
No
No
ExternalAzur
aby
aby@navithawout...
Member
No
No
navithawout
anurag
Up.anurag_gmail....
Guest
No
No
mail
Mariyam Thomas
mariyamt_cloudth...
Guest
No
No
ExternalAzur
Add or remove favorites by pressing Ctrl+Shift+F
Type here to search
Hot days ahead
11:49 AM
11/10/2025


Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
navitha_w@outlook.com
DEFAULT DIRECTORY
Home > Default Directory | Users > Users >
Create new user
Create a new internal user in your organization
User principal name *
aakash
navithawoutlook.onmic...
Domain not listed? Learn more
Mail nickname *
aakash
Derive from user principal name
Display name *
aakash
Password *
Auto-generate password
Account enabled
Review + create
Previous
Next: Properties >
Give feedback
Type here to search
Rainy days ahead
11:48 AM
11/10/2025



Entra-id connect /

Difference between entra-id and on-premise of LTIMIndtree

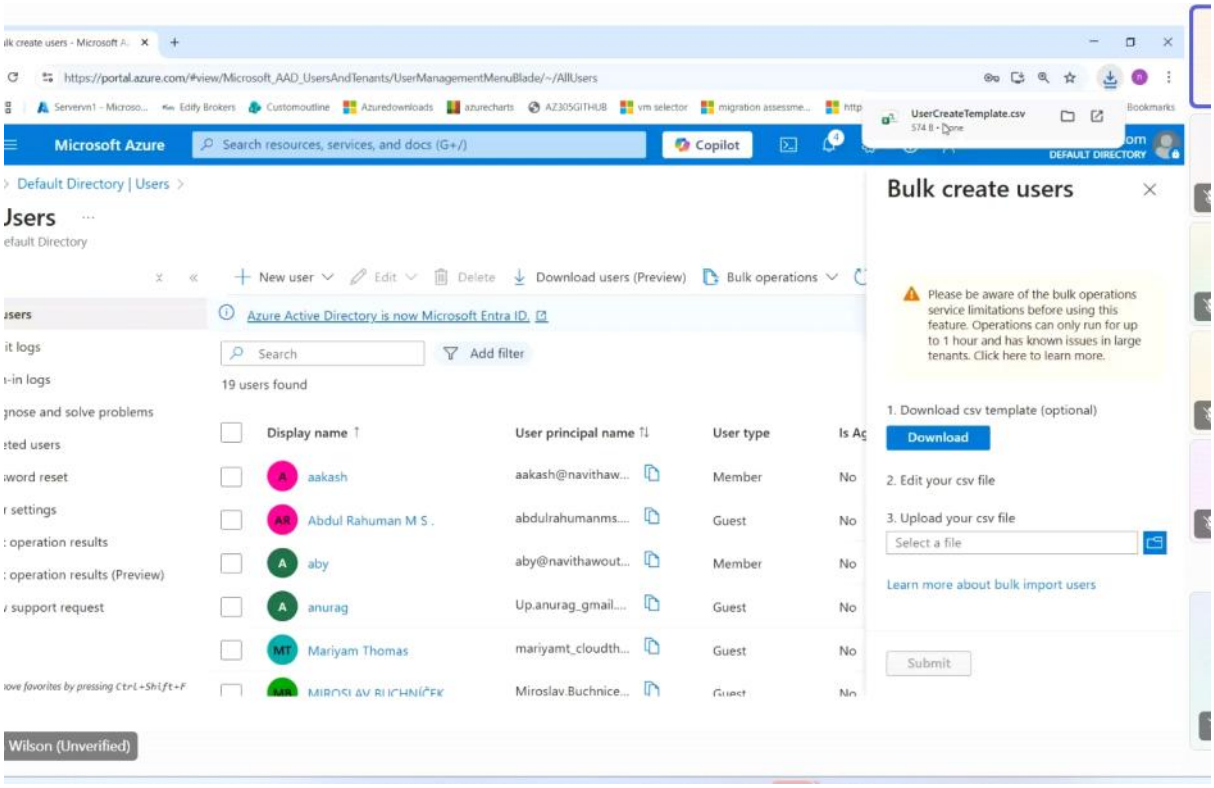


EXT added to guest user

Guest user vs member

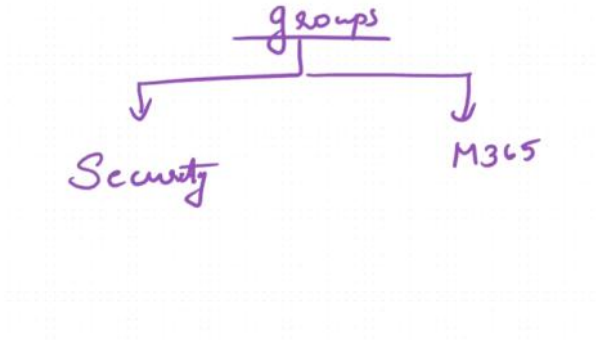
We are signing in a guest user we created Akash, and also changed password since I have admin previledges

Creating bulk users



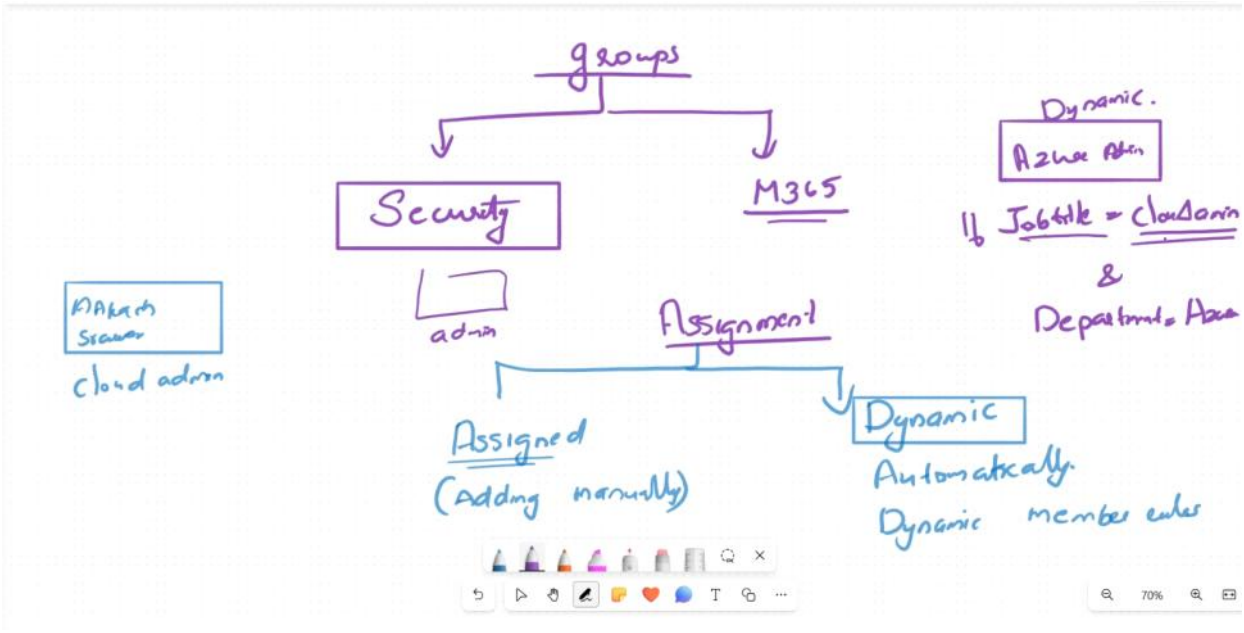
Groups

Security groups and m365 groups

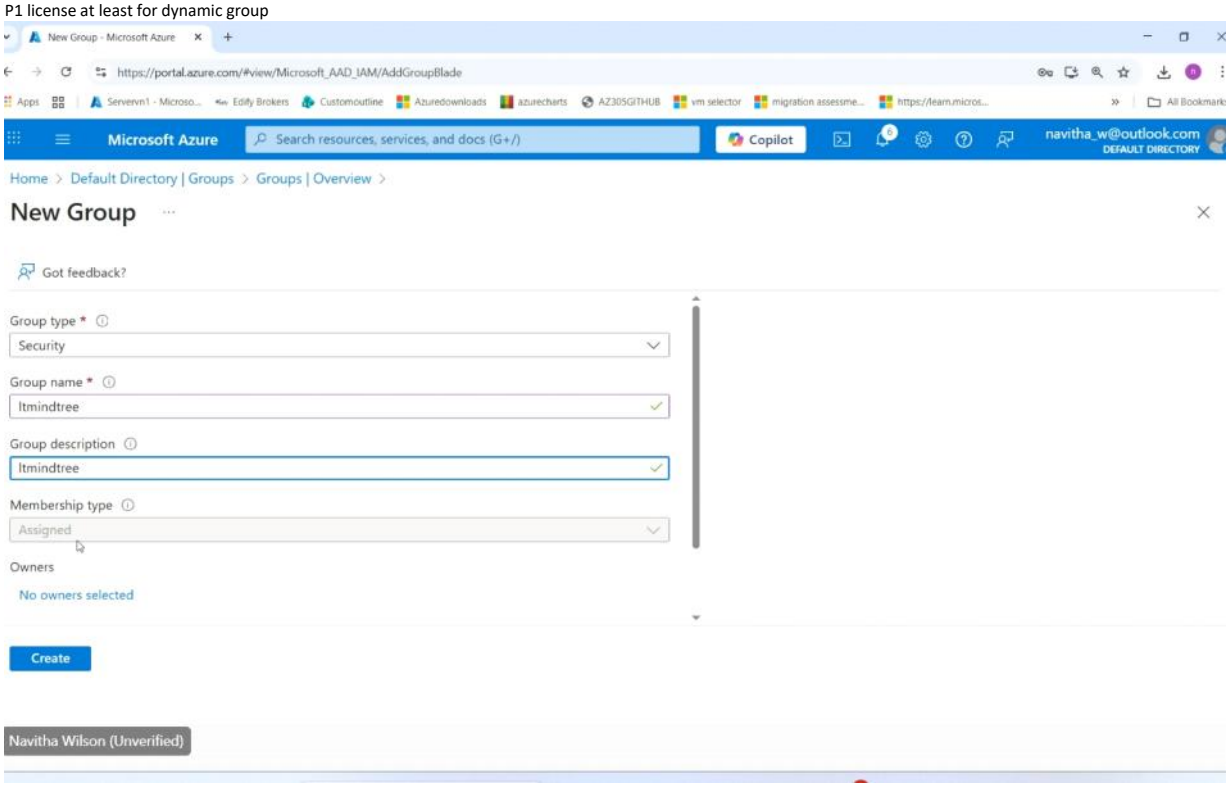


Sg -> admin group, cloud-admin group, azure-admin group -> we can assign members
We can create dynamic groups -> automatically adding based on dynmamic member rules

Suppose if job_title=cloud-admin -> all those members will be automatically added to our dynmaic group azure-admin group. We are creating dynamic rules



P1 license at least for dynamic group



Lab 01 – Manage Microsoft Entra ID Identities

In this lab, you learn about users and groups.

Users and groups are the basic building blocks for an identity solution.

You create a new user and invite a guest user.

You also create a group and add a member and owner.

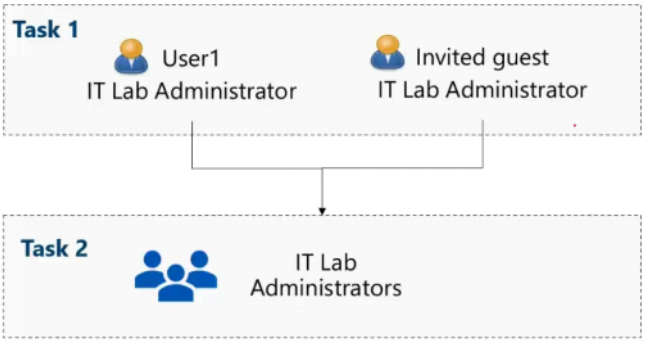
Job Skills

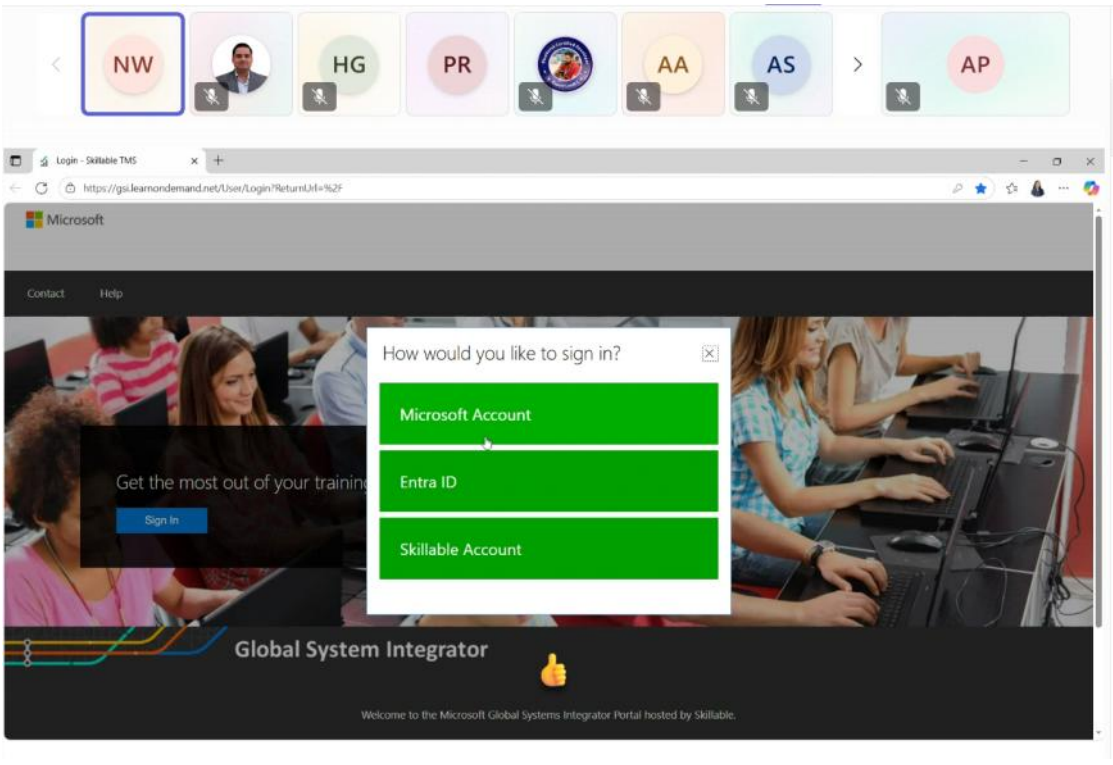
Task 1: Create and configure user accounts.

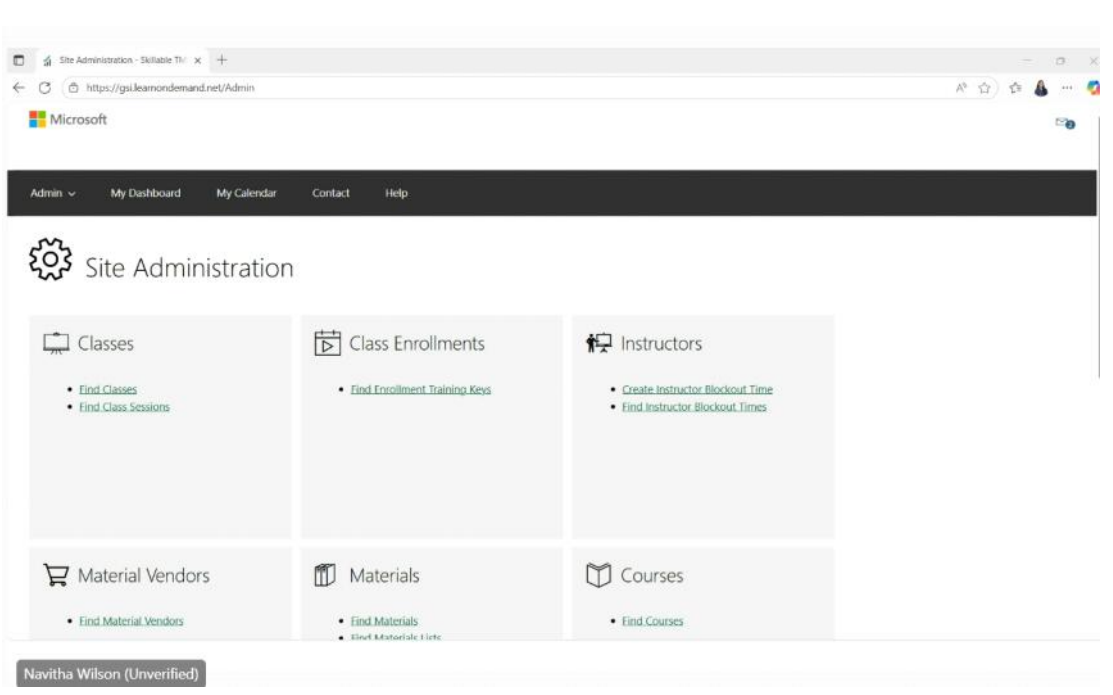
Task 2: Create groups and add members.

Next slide for an architecture diagram

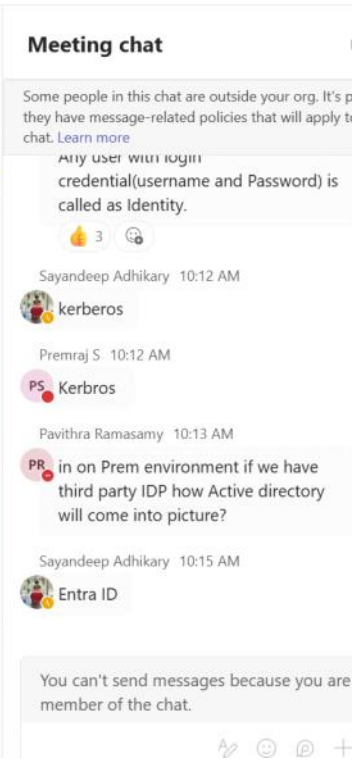
Lab 01 – Manage Entra ID Identities (architecture diagram)

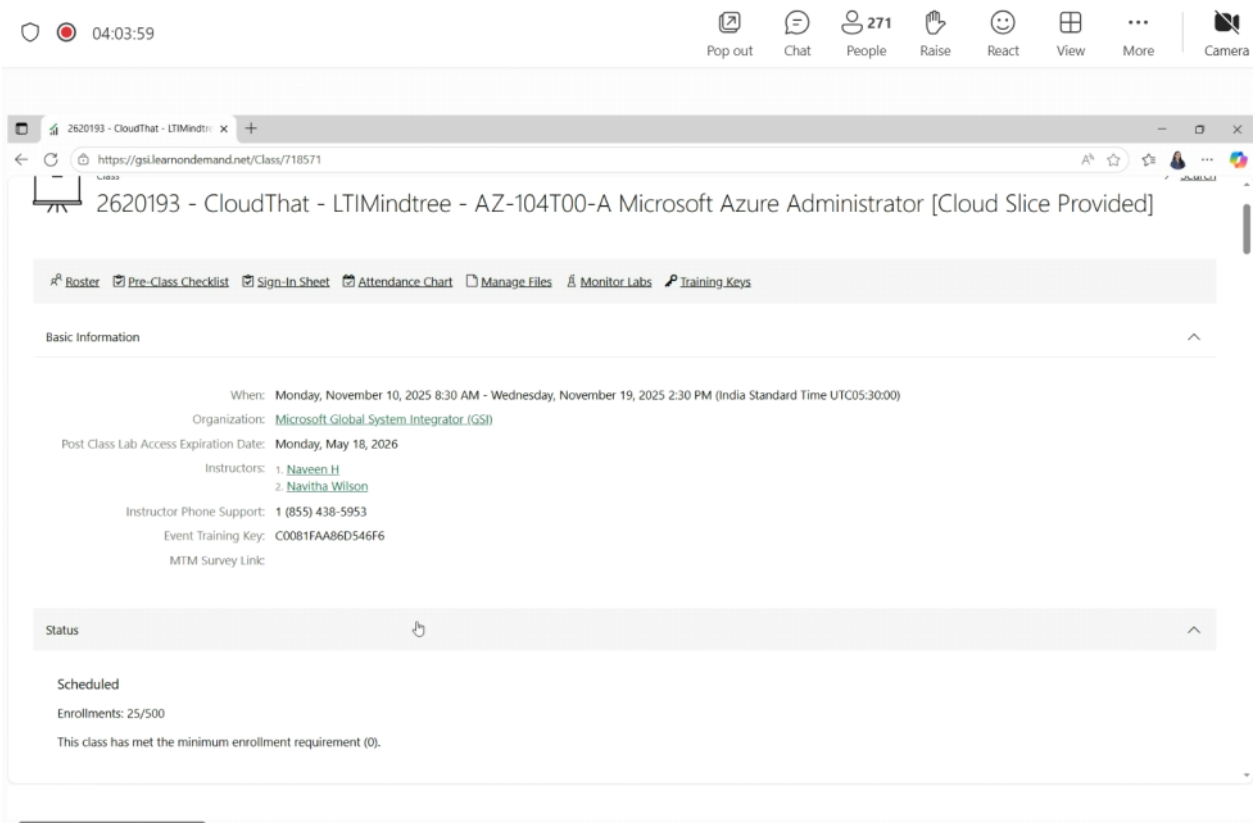




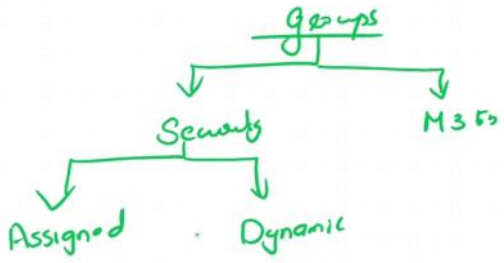


Training key





11.11.2025

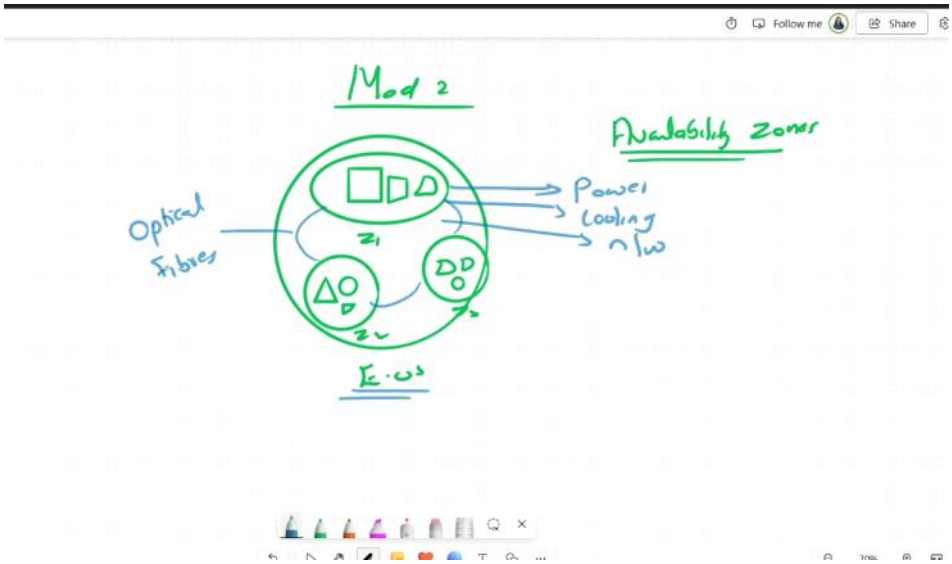


Module 2 Administrator Governance and comliance

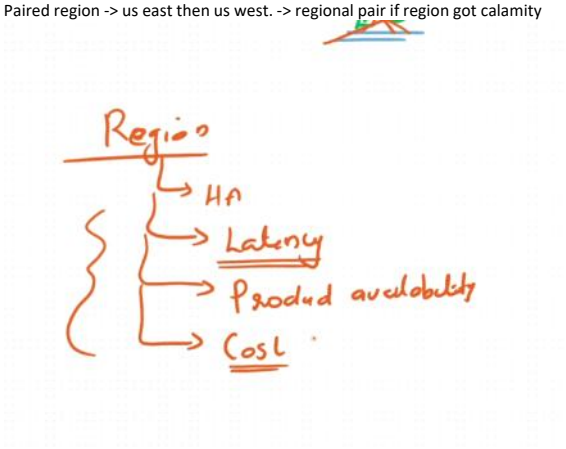
RBAC Roles for rbac features just like Entra id roles for entra id features

Hierarchy

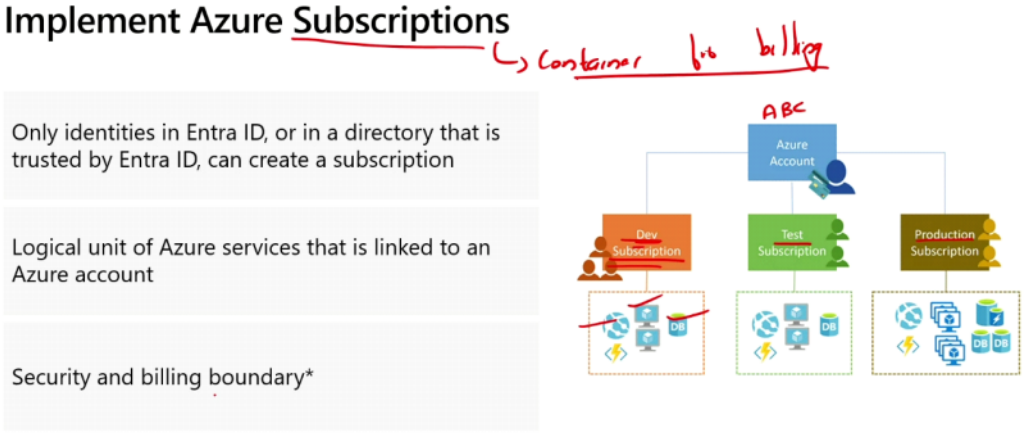
Region
Availability zones -> group of datacentres.



Paired region -> us east then us west. -> regional pair if region got calamity



Container for billing -> azure subscription



Subscription is entry point to azure resources

Identify Subscription Usage

Subscription	Usage
Free	Includes a $200 credit for the first 30 days, free limited access for 12 months
Pay-As-You-Go	Charges you monthly
CSP	Agreement with possible discounts through a Microsoft Cloud Solutions Provider Partner – typically for small to medium businesses
Enterprise	One agreement, with discounts for new licenses and Software Assurance – targeted at enterprise-scale organizations
Student	Includes $100 for 12 months – must verify student access

Resource and resource group can be in different region. because rg is containing only metadata of resource.

Resource can be of deployed only one resource group.

We can group respurce group like for vms - one rg, for storage - one rf

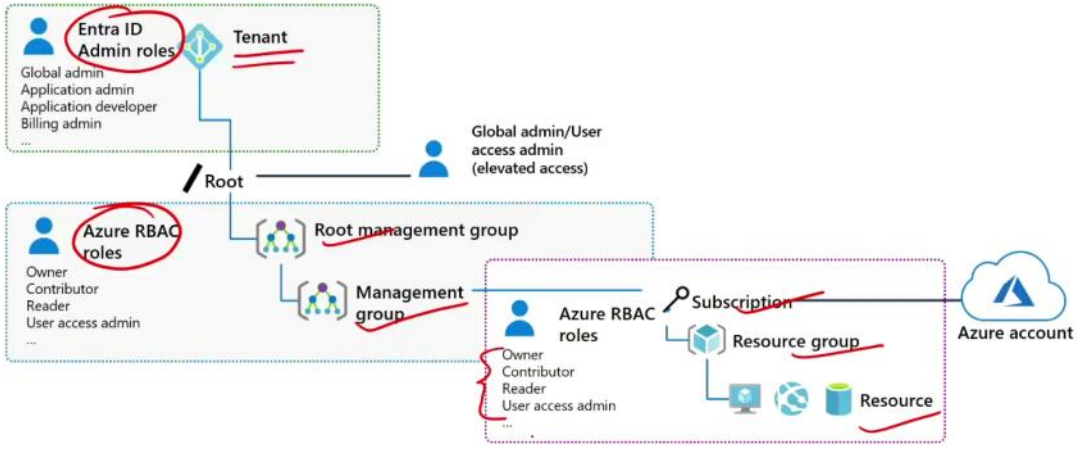
Renaming rg is not possible. rather move resource from one rg to another if needed.

if rg deleted -> all resource deleted. resource lock to avoid

locks -> delete and read only

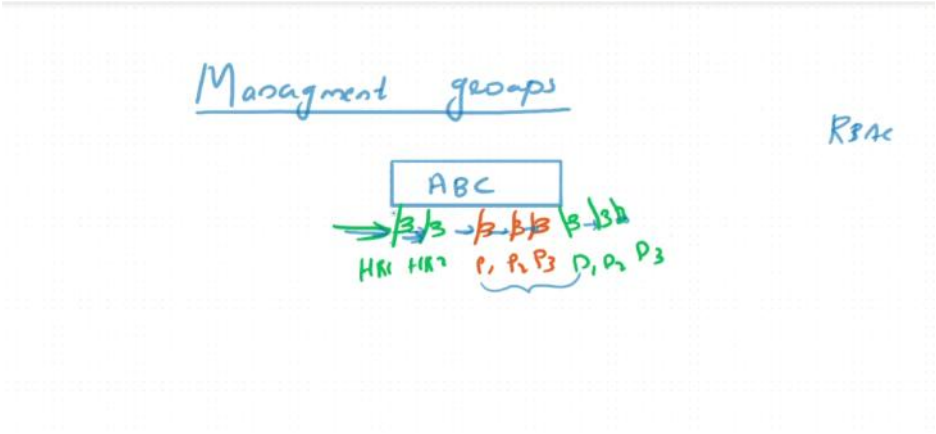
cant do any operation on resouce if read only.

Apply RBAC Authentication

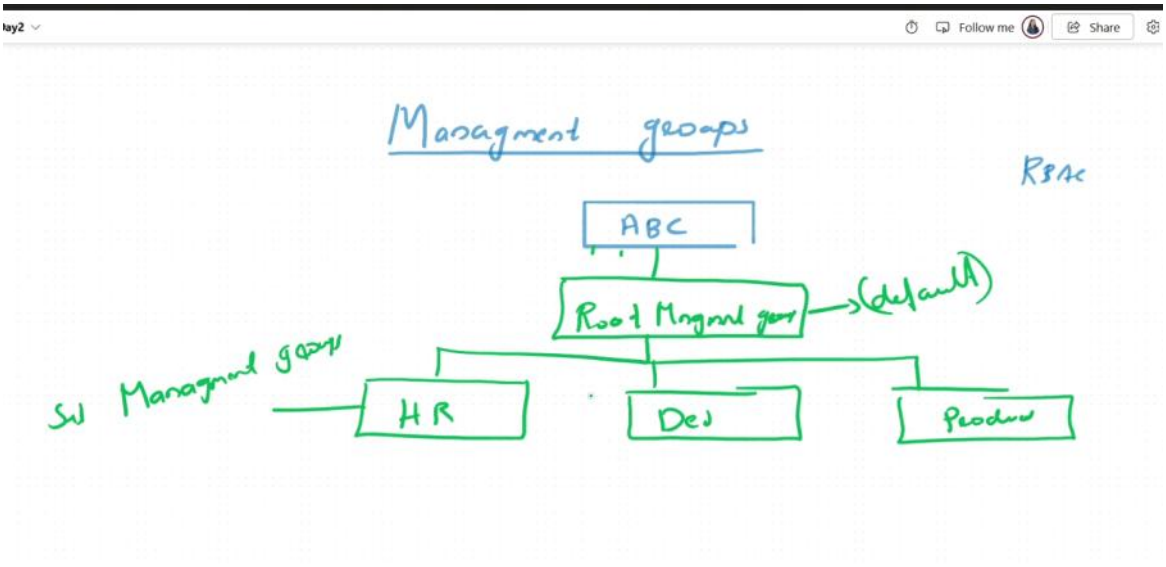


Management Groups

I need to apply policy for all subscriptions



Management groups can be used.

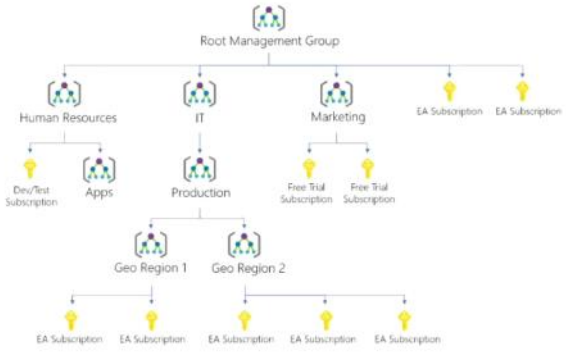


Create an Azure Resource Hierarchy

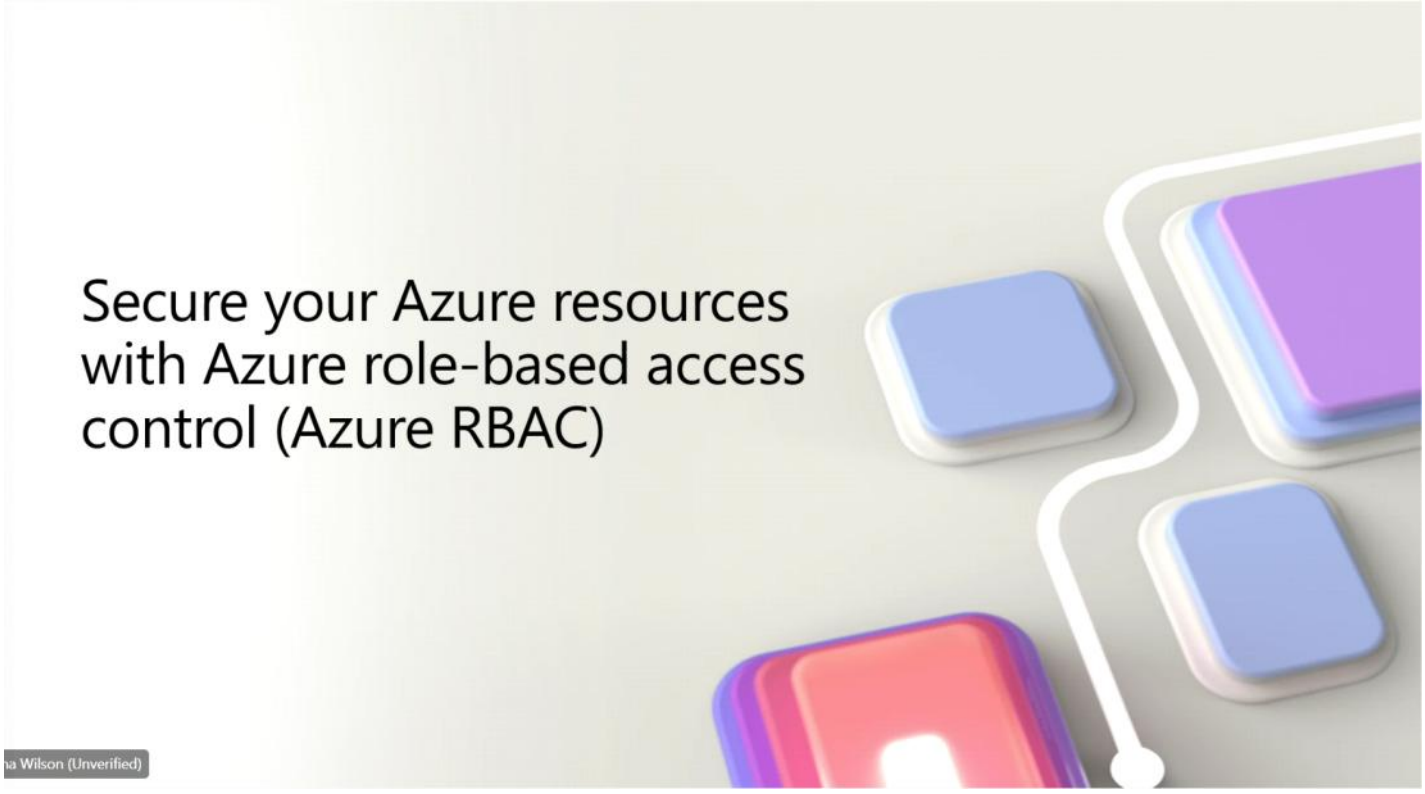
Management groups provides a level of scope above subscriptions

Target policies and spend budgets across subscriptions and inheritance down the hierarchies

Implement compliance and cost reporting by organization (business/teams)



* To prevent changes, apply resource locks at the subscription, resource group, or resources level



Create a Role Definition

Collection of permissions that lists the operations that can be performed

Lab 02a – Manage Subscriptions and Azure RBAC

In this lab, you learn about role-based access control.

You learn how to use permissions and scopes to control what actions identities can and cannot perform.

You also learn how to make subscription management easier by using management groups.

Job Skills

Task 1: Implement management groups.

Task 2: Review and assign a built-in Azure role.

Task 3: Create a custom RBAC role.

Task 4: Monitor role assignments with the Activity Log.

Lab 02b – Manage Governance via Azure Policy

In this lab, you learn how to implement your organization's governance plans.

You learn how Azure policies can ensure operational decisions are enforced across the organization.

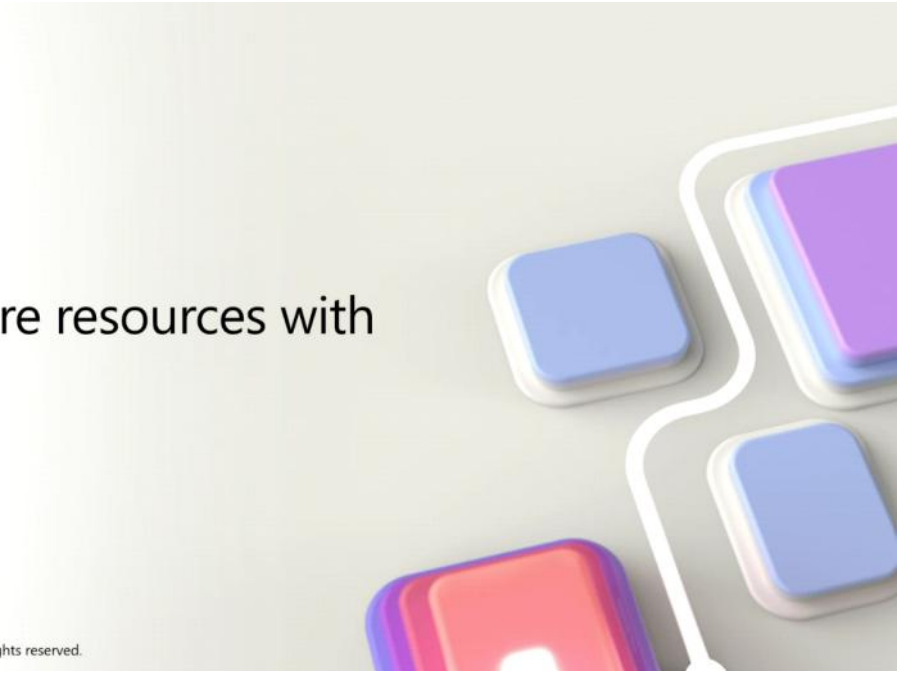
You learn how to use resource tagging to improve reporting.

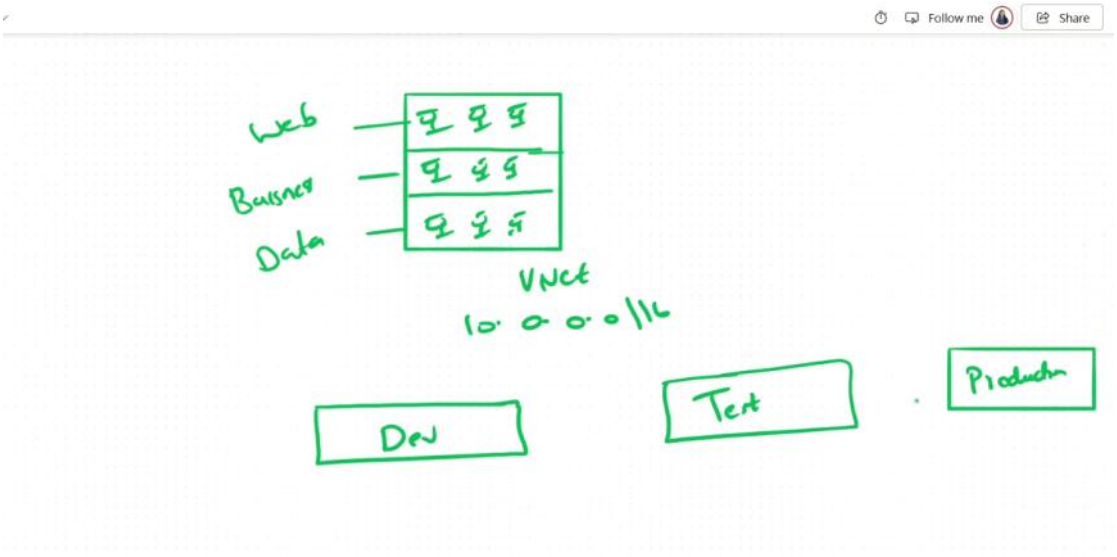
Job Skills

Task 1: Create and assign tags via the Azure portal.
Task 2: Enforce tagging via an Azure Policy.
Task 3: Apply tagging via an Azure Policy.
Task 4: Configure and test resource locks.

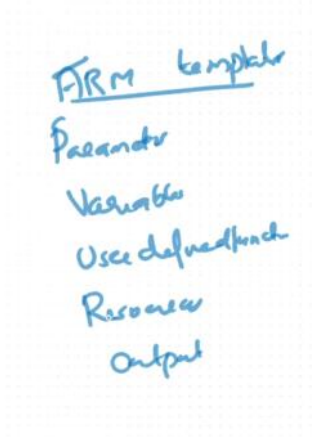
Next slide for an architecture diagram

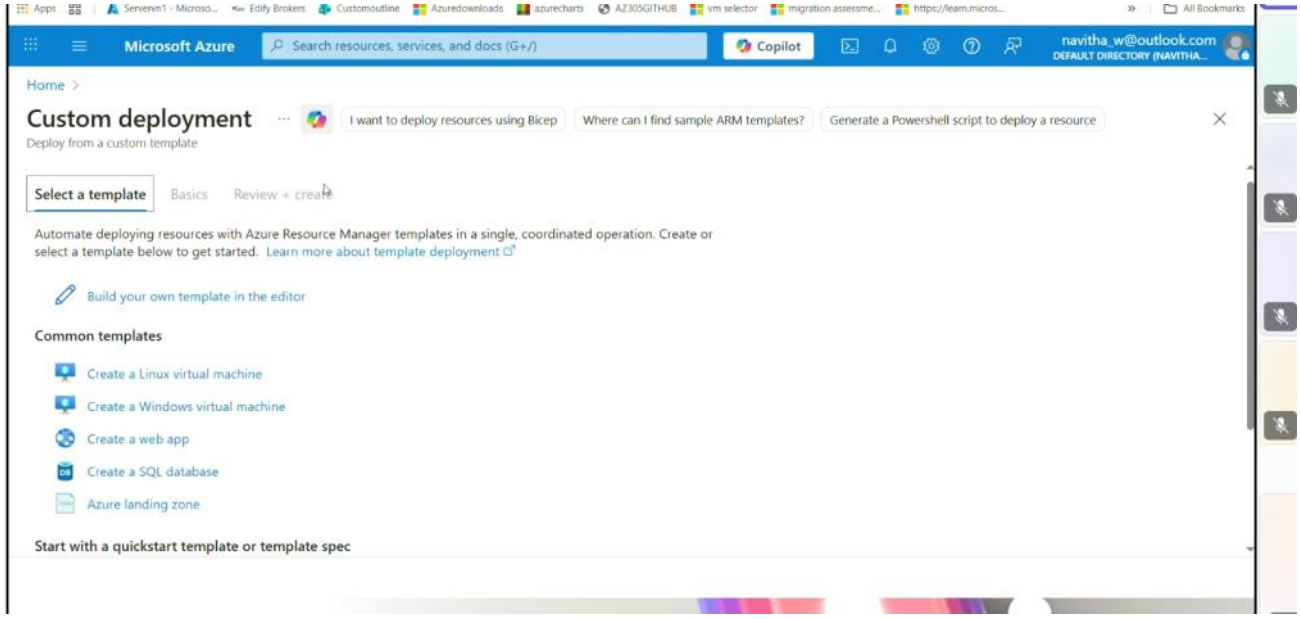
Deploy Azure resources with templates





Code as a service
ARM Template





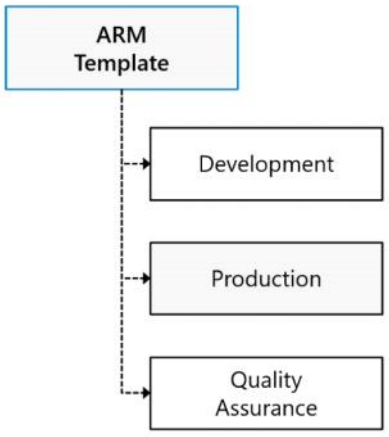
Custom template for testing, prod and develoent

Parameter file

Deploy vm through custom template method

Review ARM Template Advantages

- Improves consistency and promotes reuse
- Reduce manual, error prone, and repetitive tasks
- Express complex deployments
- Express requirements through code
- Provides validation tasks
- Modular and can be linked
- Simplifies orchestration



Explore the JSON Template Schema

- Defines all the Resource manager resources in a deployment
- Written in JSON
- A collection of key-value pairs
- Each key is a string
- Each value can be a string, number, Boolean expression, list of values, object

```
{
    "$schema":
      "http://schema.management.
      azure.com/schemas/2019-04-
      01/deploymentTemplate.json#",
    "contentVersion": "",
    "parameters": {},
    "variables": {},
    "functions": [],
    "resources": [],
    "outputs": {}
}
```

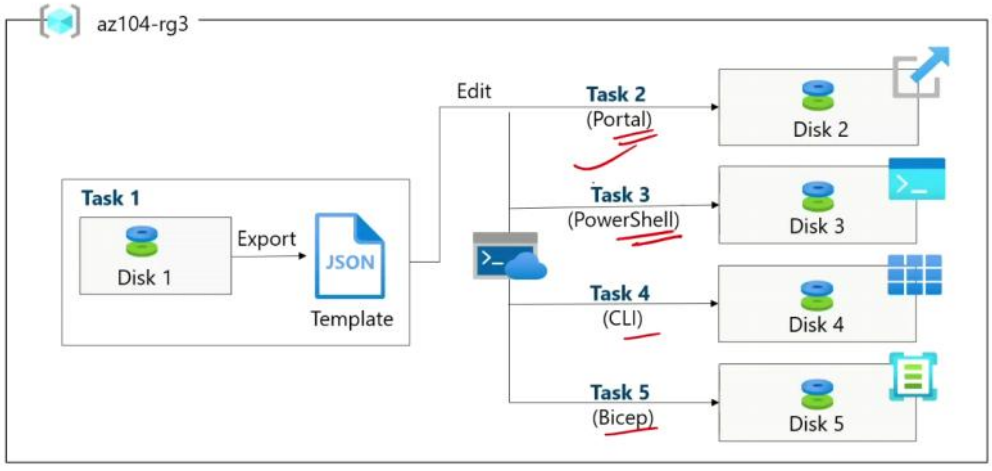
Consider Azure Bicep Files

- Simpler syntax for writing templates
- Smaller module files you can reference from a main template
- Automatically detect dependencies between your resources
- Visual Studio Code extension with validation and IntelliSense

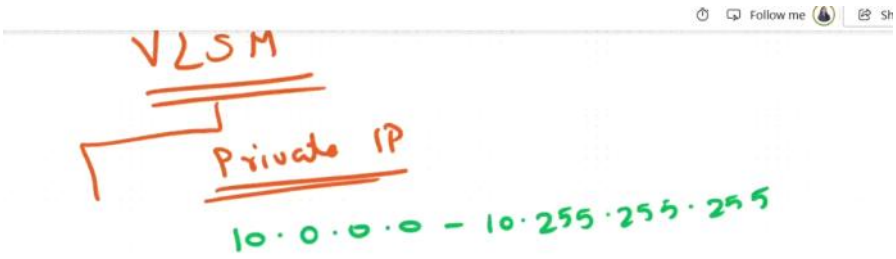
Bicep file

```
resource storageAccount
'Microsoft.Storage/storageAccounts@
2021-01-01' = {
  name: storageAccountName
  location: location
  tags: {
    displayName: storageAccountName
  }
  kind: 'StorageV2'
  sku: {
    name: 'Standard_LRS'
  } }
```

Lab 03 – Architecture diagram

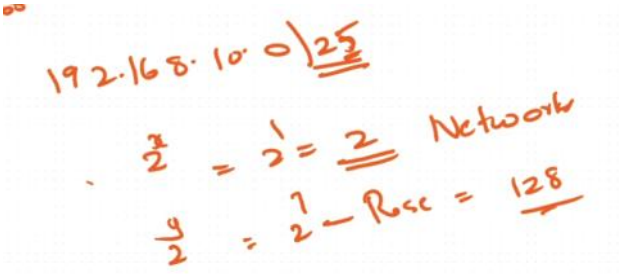


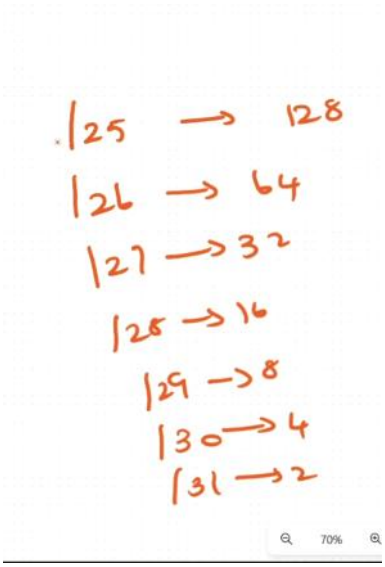
Subnet masking
VLSM

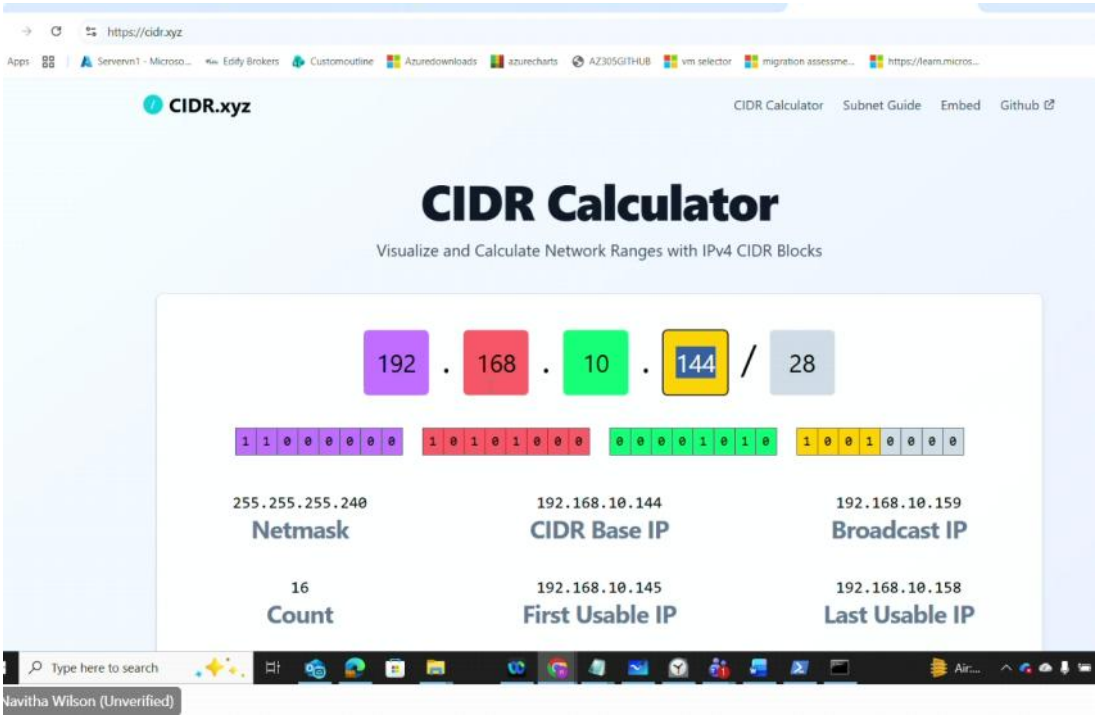


/24 notation

Borrowing bits from host -> /25 -> VLSM -> to avoid wastage







Management groups are used to logically organize and segment subscriptions.

For example, if your organization has a dedicated support team for Europe, you can organize European subscriptions into a management group to provide the support staff access to those subscriptions (without providing individual access to all subscriptions).

Azure Vnet -> software defined.

regional service
pvt

NSG -> allows Subnet to exclude connecting with the other subnet in vnet.

Great question! **Application Security Groups (ASGs) are applied at the NIC (Network Interface Card) level**, not at the subnet level.
Here's why:

- Each **VM in Azure has a NIC**, and that NIC is what connects the VM to the virtual network.
- When you assign an ASG, you **associate it with the NIC of the VM**, meaning the VM becomes part of that ASG.
- **NSG rules** then reference ASGs as the **source or destination** for traffic filtering.

Key Points

- **ASG ≠ Subnet**: ASGs do not apply to entire subnets; they apply to individual NICs.
- You can have multiple ASGs on a single NIC.
- This gives **fine-grained control** over which VMs belong to which security group, even if they share the same subnet.

☑ **Example**:

- Subnet: 10.0.0.0/24
- VMs: VM1, VM2, VM3
- Assign ASG WebServers to NIC of VM1 and VM2.
- Assign ASG DBServers to NIC of VM3.
- NSG rule: Allow traffic from WebServers to DBServers on port 1433.

Do you want me to **create a visual diagram** showing how ASG attaches to NIC and how NSG rules reference ASGs? Or should I give you a **step-by-step Azure Portal setup guide**?