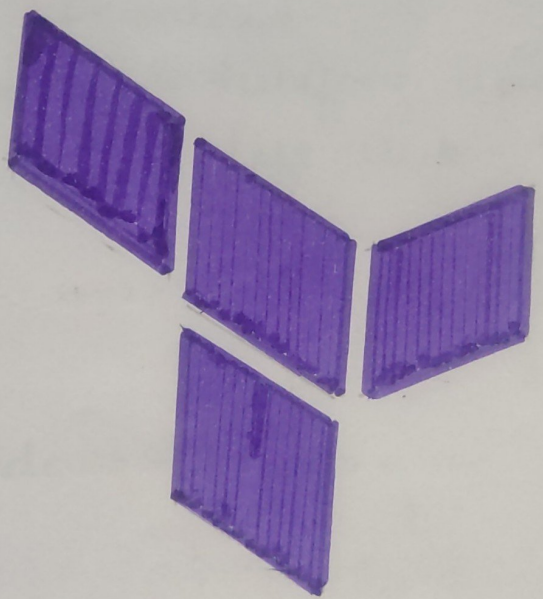
TERRAFORM



* Terraform - It is a infrastructure as code tool
  - used to automate.
  - open source not in 2023

- EC2 create virtual server
- VPC helps in network

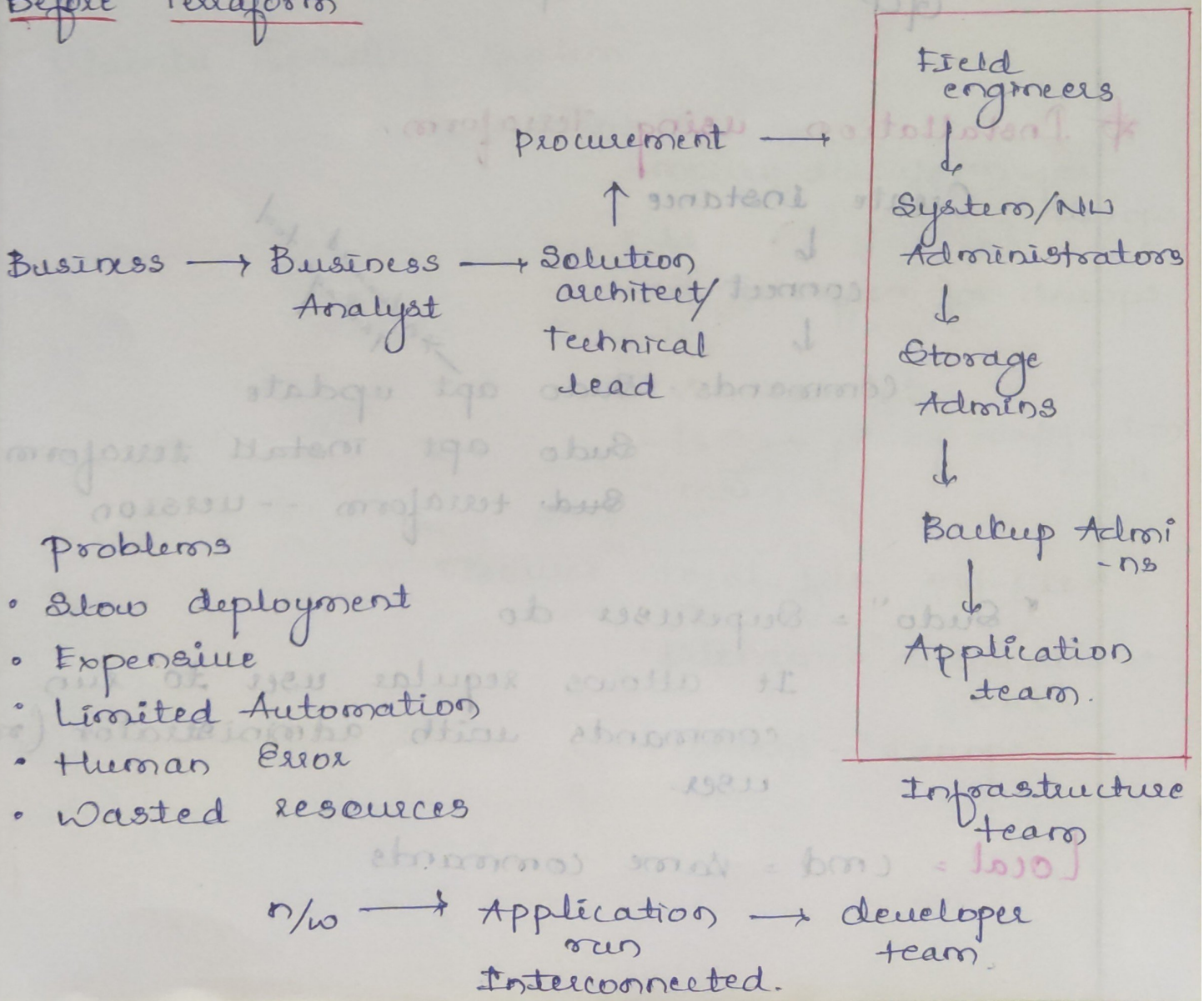
} to automate this we write a code that is IAC

- Mitchell Hashimoto: created the tool terraform in 2014

problem = Creating infrastructure manually was the biggest task, so terraform came into picture.

- Terraform became business source = 2023 → License

- Before Terraform



Problems
- Slow deployment
- Expensive
- Limited Automation
- Human Error
- Wasted resources

* Why terraform is used?
→ For automation, resources should not waste.

* Difference.
- Terraform is IAC tool which is provisioned to private & public cloud.
- Ansible is IAC tool which is used for CFM (configuration management)

Infrastructure (Resources/Services)

AWS	EC2	EKS	} Terraform.
Azure	Blob	AKS	
GCP	"	GKS	

* Installation using terraform

Create instance
↓
connect
↓
Commands : Sudo apt update → advanced tool packaging
Sudo apt install terraform -y
Sud terraform --version

"Sudo" = Superuser do
It allows regular user to run commands with administrator (root) user

Local = cmd = Same commands

* Teraform HCL (Hashicorp Configuration Language)

<block> <parameters> {

= arguments

}

Through teraform we can create infrastructure

Block = resources
        variable
        Output

arguments
instance = myserv
teppe = t2.micro

resources = S3 bucket, EKS,
parameters = aws_instance my-instance

ubuntu Operating System =

ubuntu@ip = ls
= mkdir terraform-for-devops
= ls
= cd terraform-for-devops
= ls
= vim new.txt
= ls
= main.tf → primary configuration file

resource local_file my-file {
filename = "automation.txt"
Content = "Devops"
}

- When we do 'ls' we can see nexo_txt, main.tf file
  where is automated.txt file

- I created main.tf file but terraform dont know about that file

"terraform init"

↓

terraform environment mil gaya

After "init" command we cannot see the environment

Command = ls -a → all

↓

Shows the hidden file

- The file I created is valid or not

"terraform validate"

terraform init

" validate

Provides the O/p

↑

" plan → Terraform apply

↓

It shows how the O/p looks

* terraform plan = used to preview the changes terraform will make to your infras -tructure based on the current configuration file (.tf)

- It helps you see what will be created, modified or destroyed before actually applying any changes.

* Commands = terraform apply

ls

(automate.txt main.tf new.txt terraform.tfstate)

cat automate.txt

(best content for devops)

* Block = A block is a container for configuration. It defines a type of resource or setting & may contain arguments or other block inside it

```
resource "aws_instance" "web" {
    ami = "ami-123"
    instance_type = "t2.micro"
}
```

resource is a blocktype

"aws_instance" "web" = its labels

* Argument = It is a key-value pair within a block that configures a specific setting for that block.

Ex: ami = "ami-123"
instance_type = "t2.micro"

These arguments inside the aws_instance block. They tell terraform which AMI to use & what instance type to launch.

* **Attribute :**

An attribute is a piece of data that is either configurable (like arguments) or read-only & exported after resource creation.

Ex = output "instance_ip" {
value = aws_instance.web.public_ip
}

public_ip = attribute of the aws_instance resource.

You didn't set manually - it was generated by AWS & made available by terraform.

Command = "terraform destroy"
deletes everything

"terraform -auto-approve"
↓
It doesn't ask for the question yes/no to delete.

Providers

block ← resource local_file my_file {

parameters → filename = "automate.txt"

content = "best content for devops - TWS"

resource type → local_file

resource name / identifier → my_file

}

providers - plugin that interact with the services, cloud providers & other APIs.

Commands = ls -a

(terraform)

(terraform.lock.hcl)

↓

cd .terraform/

↓

ls (providers)

↓

cd providers/

↓

ls (registry.terraform.io)

↓

cd "/"

↓

ls (hashicorp)

↓

cd hashicorp/

↓

ls (local)

* Chrome = terraform providers = local = documentation = resources = local.txt

* Create S3 bucket using terraform.

```
resource aws_s3_bucket my-bucket {
    bucket = "aws terraform in one shot"
}
```

+ We don't have AWS providers so once again we will do "terraform init"
  default provider will be there,
  we need to install aws provider

* Steps to install terraform aws provider.

Chrome → aws terraform provider → use provider → copy & paste in your terraform configuration

↳ Same with azure provider

↑ block

no parameters

```
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
            version = "5.91.0"
        }
    }
}
```

:wq

Change directory: cd ./downloads/

terraform init = once again should be done for aws providers

when we repeat these steps again

cd .terraform/
cd providers
cd registory
cd hashicorp
Here we can see (local, aws)

* aws cli - to connect

Chrome
↓
aws cli
↓
update / install
↓
Linux ⟶ Same with windows.
↓
copy & paste in terminal.
(curl install)
↓
terminal = ls
(awscliv2.zip terraform)
↓
to unzip this
↓
terminal : Sudo apt-get install unzip
↓
unzip awscliv2.zip
↓
aws --version
⇓
check if won't get
(documentation)

Command = aws configure

↓

Access key = ?

↓

to know this create IAM.

↓

user

↓

S3 full access

↓

Created user

↓

Security Credentials

↓

Create Access key.

↓

copy & paste in terminal

Command = "aws s3 ls"

↓

it shows all the S3 buckets I created on my account

to be in same region

* How to configure providers in terraform.

cd terraform

↓

vim provider.tf

↓

provider "aws" {

region = "us-east-2"

}

↓

terraform plan

↓

terraform apply

S3-bucket will be created automatically in aws

* VS - Code :→

terminal = aws configure

Access key = copy & paste

"

Chrome = aws terraform provider

VS code = file → terraform.tf → c/p

VScode → file → provider.tf → c/p

region = "ap-south-1"

Create ec2 using terraform:

requirements = key pair (login)

VPC & security group

ec2 instance

1) To generate a keypair in VS code

Command = `ssh-keygen`

provide name = terra-key-ec2

Enter

Enter

weget private & public key

terra_key-ec2 = private

terra-key-ec2.pub = public key.

ec2.tf

```
resource aws_key_pair my_key {
    key_name = "terra_key_ec2"
    public_key = file("terra-key-ec2.pub")
}
```

or directly paste Public key

Q) VPC

```
resource aws_default_vpc default {

}

resource aws_security_group my-security-group {
  name = "automate-sg"
  description = "this will add a tf generated security group"
  vpc_id = aws_default_vpc.default.id
}
```

↓ continue

interpolation (Extract values)

* Interpolation = inserting the dynamic values like variables / resource into strings

```
#inbound rules [22 & 80]
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0]
    description = "SSH open"
```

~~outbound rules~~

```
egress {
        from_port = 80
        to_port = 80
        protocol = "TCP"
        cidr_blocks = ["0.0.0.0/0"]
        description = "HTTP open"
    }
ingress {  ⟶ this is same as how we edit inbound rules in AWS.
        from_port = 8000
        to_port = 8000
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
        description = "flask app"
    }

# outbound rules
    egress {
        from_port = 0 ⇒ all port
        to_port = 0
        protocol = "-1" [covers all port]
        cidr_blocks = ["0.0.0.0/0"]
        description = "all access open outbo
                                    -und"
    }

  tags = {
      Name = "automate-sg"
    }
}
```

EC2 instance

```
resource "aws_instance" "my-instance" {
    key_name = aws_key_pair.my-key.
               key_name
    security_groups = [aws_security_group.
                       my_security_group.name]
    instance_type = "t2.micro"
    ami = "ami-0cb912ebcad"
```

copy from ec2 → ami → ubuntu

```
    root_block_device {
        volume_size = 15
        volume_type = "gp3"
    }
    tags = {
        Name = "tws_automate"
    }
}
```

Command: terraform init

" validate

plan

apply

S3 - full access

administrator full access

terraform apply

"EC2 gets created"

ssh connect = Same copy & paste in vs code terminal

TO destroy = terraform destroy -auto-approve

* Variables :→ (Variable.tf)

putting fixed values directly into your code

Instead of hardcoding values like AMI_ID, instance type or region, you define them as variables & reuse them anywhere in your code.

```
variable "ec2_instance_type" {
    default = "t2.micro"
    type = string
}

variable "ec2_root_storage_size" {
    default = 10
    type = number
}

variable "ec2_ami_id" {
    default = "ami-0cb91c7de36eed2cb"
    type = string
}
```

Instead of doing ssh -i to connect we can use output block

output.tf

```
output "ec2_public_ip" {
    value = aws_instance.my_instance.
            public_ip
}
```

```
output "ec2-public-dns" {
    value = aws.instance.my-instance.public
            -dns
}
output "ec2-private-ip" {
    value = aws_instance.my-instance.
            private_ip
}
```

terminal - terraform plan

At a time run the instance & install nginx.

/* first destroy then run once again

ec2.tf - modify:

```
user_data = file("install-nginx.sh")
```

install_nginx.sh (file)

```
Sudo apt-get update
Sudo apt-get install nginx
Sudo systemctl ~~start~~ "
  "   "    enable "
```

```
echo "<h1> Terraform in one shot by tws </h1>"
| Sudo tee /var/www/
```

* primary use of HCL - to define resources & configuration in a human read-able format
* command creates a execution plan: terraform plan

* Default file created by terraform to store the execution plan = plan.out

* .terraform directory contain = provider plugin & metadata

* valid provider form: provider "aws" {
  region = " "
}

* Terraform file typically holds the resource configuration. = main.tf

* Terraform apply = Executes the changes ~~ap~~outlined in the plan.

* IAM used in conjunction with terraform for AWS = TO provide credentials for accessing AWS resource

* terraform fmt = reformats configuration files to canonical HCL style

* terraform command to verify the correctness of configuration files = terraform validate

* Correct way to import an existing AWS S3 bucket into terraform = use the terraform import command with the resource & bucket name.

* --target option do when used with the terraform destroy command = It destroys only the specified resource & its dependencies

Ex: terraform destroy --target = aws_instance.my_instance.

↓

it destroys the instance but not the key pair, VPC ....

* purpose of root_block_device in a terraform resource = To define the configuration of the root storage volume for an instance

* LOOPS & CONDITIONS :→

* Arguments ; Loops - [for-each , count]

"count" = meta argument

it creates the multiple resources as per requirements

Eg: resource "aws_instance" {

count = 2 → it makes 2 instances

:

}

output.tf = The file we created

It provides the error as it won't support multiple instance

modify =

output.tf

output "ec2_public_ip" {

value = aws_instance.my_instance[*].public_ip

↓

count

" public_dns

" private_ip

* Types of instances :→
  Spot instance ⟶ bidding - they have 500 instances free
  on-demand instance
  ↓
  they will provide it in less price

"for_each" = meta keyword.
  ↳ argument
  ↓
  Create multiple resources from a list/map

- Instead of writing the same block again & again, you let terraform loop through values

Ex = count = 2
  ↳ it create same instances
for_each = it makes different instances at a time

```
for_each = tomap ({
  ⋮
})
```
} get 3-instance get created.

```
for_each = tomap ({
  "tws-instance-micro" = "t2.micro"
  "tws-instance-medium" = "t2.medium"
})
```
2 instance

# to access another key for instance we use
name = each.key
location = each.value

the same with
private ip, public dns ↑ ec2.tf.

```
output "ec2-public-ip" {
    value = [
        for instance in aws_instance.
            my-instance : instance.publi
                                -ip
    ]
}
```

terminal = terminal plan.

* depends_on ⇒ meta argument

ec2.tf

```
depends_on = [aws_security-group.my-
              security-group , aws]
```

It helps when terraform can't configure out the order by itself.

Ex: you want to launch an EC2_instance, but only after a Security group is created
↓
dependent

1. Terraform will create the Security group 1st
2. Then launch the Instance
3. depends_on (Extra control)

↓

first creates the dependencies

Ex VPC
SG
Instance

trick = if environment production
volume = 20
else
volume = 10

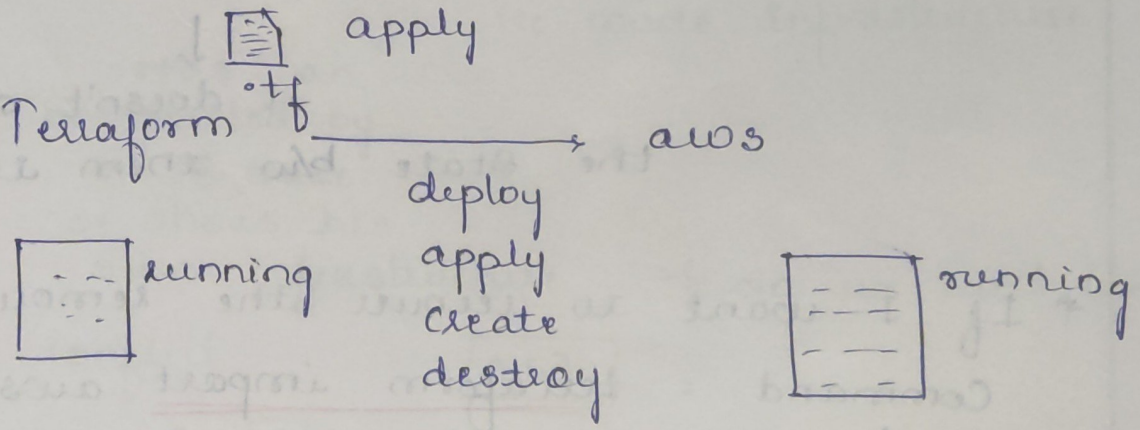
ec2.tf

```
root_block_device {
    volume_size = var.env == "prd" ? 20 : var.ec2
    volume_type = "gp3"                - default_root
}                                      - storage-size
```

↓
Conditional Statement / ternary Statement

* Terraform State management & backends :→

* State :→



Ex: If the instance is running in aws the terraform should know that the instance is running in AWS ⇒ so this thing is only called as = "State"

"Tf State" = In terraform has a ~~be~~ create file it ensures that everything running correctly maintain the files.

* Terraform state - Terraform keeps track of all the infrastructure it creates in a file called terraform.tfstate

click "terraform.tfstate"

Comma-nd : terraform refresh.

terraform state list
: aws_instance
: aws_key_pair
...

terraform show aws_key_pair

* In case if I don't want to maintain the some resources Ex: aws_key_pair

Command : terraform state rm aws_key_pair

↓

it doesn't delete anything the state b/w them is removed.

* If I want to recover the removed file

Command : terraform import aws_key_pair.my-key key-0d156aaf3d...

↓ ec2.tf

↓

copy paste from ec2 you created through terraform.

```
resource "aws_instance" "my-new-instance" {
    key_name = "unknown"
    public_key = " ... "
}
```

terminal = terraform import aws_instance.my-new-instance i-0fs...

↓

c/p from created instance

we can access the manual created instance : Ex "terra instance".

* **problems** :→ (terraform state management)

1) tf state file cannot be accessed or commit to Github →

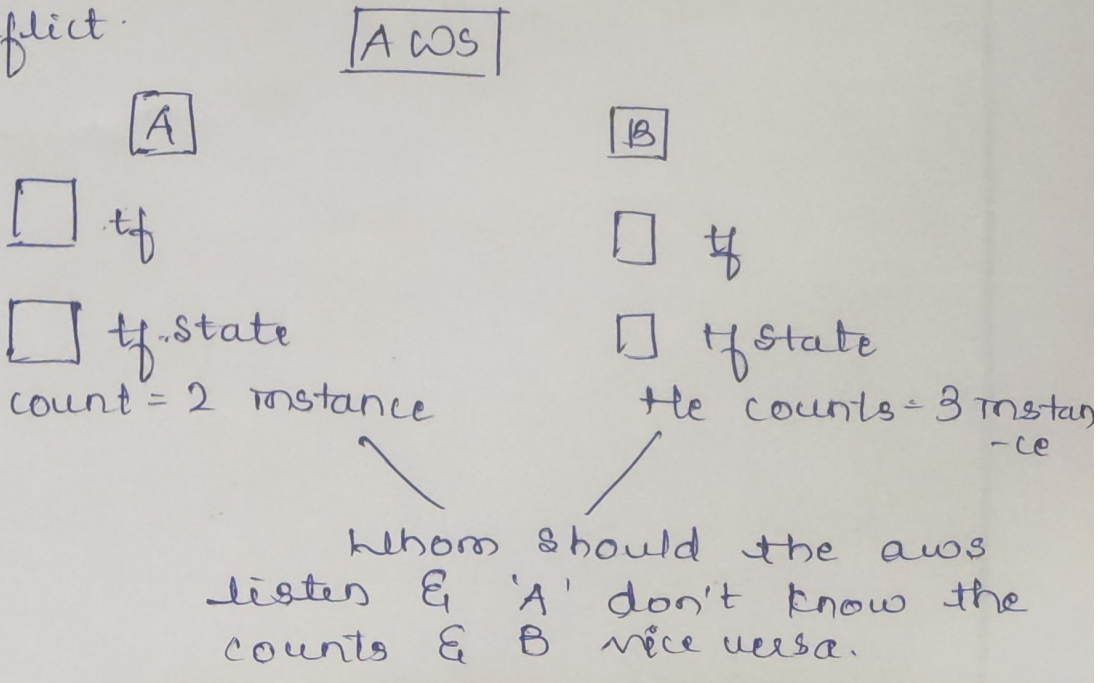
2) reason is

Im A another B

He made infrastructure

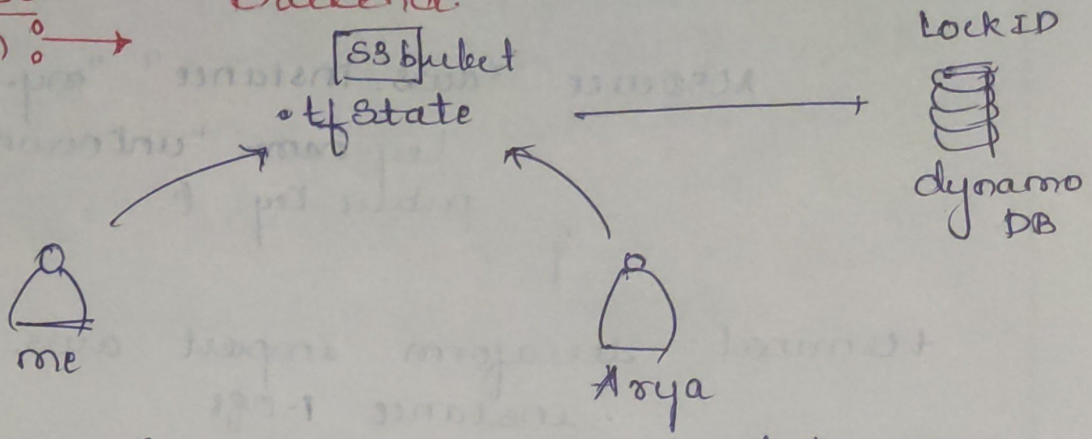
this can be pulled by 'A' & make or shows his own infrastructure ⇒ so we can't commit

2) State conflict.



Solution = tf state file → we require S3 & dynamoDB

* Lock & release Mechanism :→ Backend.



If arya is accessing the bucket the S3 - triggers & send it to the dynamo DB a lock id is generated

Arya created the lock ID until that lock ID is not ~~accessed~~ removed other users cannot access it.

Till the lock ID is there no one can access it.