Terraform can help you manage in a cloud infrastructure setup:

**1. Compute Resources:**

-> Servers (EC2, Compute Engine, etc.)

-> Auto-scaling Groups

-> Instances with specific configurations (e.g., AMI, instance type, etc.)

**2. Networking:**

-> VPC (Virtual Private Cloud)

-> Subnets (Public, Private)

-> Route53 (DNS management)

-> Internet Gateways, NAT Gateways

-> Virtual Private Gateways (VPN)

**3. Security & Identity Management:**

-> Security Group

-> IAM (Identity and Access Management)

-> Roles, Policies, and Permissions

-> Service accounts, and Access management

-> Users and Groups

-> Managing user access with policies

-> Key Pairs (for EC2 instances)

**4. Load Balancers & Auto Scaling:**

-> Elastic Load Balancers (ELB)

-> Application Load Balancers (ALB)

-> Network Load Balancers (NLB)

-> Target Groups and Listeners

**5. Databases:**

-> RDS (Relational Database Service)

-> DynamoDB (NoSQL)

-> Aurora

-> Elasticache (Redis, Memcached)

**6. Storage:**

-> S3 Buckets

-> EBS Volumes

-> EFS (Elastic File System)

-> Glacier Storage

**7. Monitoring & Logging:**

-> CloudWatch (Metrics, Alarms, Logs)

-> CloudTrail (Audit Logs)

-> AWS Config (Compliance and Change Tracking)

**8. Serverless Resources:**

-> Lambda Functions

-> API Gateway

-> Step Functions

-> SNS (Simple Notification Service)

-> SQS (Simple Queue Service)

**9. Containerization**:

-> ECS (Elastic Container Service)

-> EKS (Elastic Kubernetes Service)

-> Fargate

-> Docker Containers & Tasks

**10. CI/CD Pipeline Resources:**

-> CodePipeline

-> CodeBuild

-> CodeDeploy

**11. Infrastructure as Code:**

-> Terraform Modules

-> Reusable infrastructure code for different services

-> State Management

-> Backend state storage (e.g., S3, Terraform Cloud)

**12. Other Cloud Providers:**

-> Azure, GCP (Terraform supports multi-cloud infrastructures)

-> Google Cloud Resources (GKE, Compute Engine, Cloud Functions)

-> Azure Resources (VMs, App Services, SQL Database)

**13. Miscellaneous Resources:**

-> Secrets Management (e.g., Secrets Manager, KMS)

-> CloudFront (CDN)

-> API Gateway

Terraform is extremely versatile for automating and provisioning infrastructure. You can also leverage modules, workspaces, and other features to make your infrastructure code more modular and maintainable.

**Terraform Advantages:**

**-> Version Control:**

Terraform is code, and like any code, it should be kept in version control (e.g., Git). This allows you to track changes over time and easily revert to previous versions if something goes wrong. It's like having a "history" of your infrastructure.

**-> Consistent Infrastructure:**

With Terraform, the same code can be used across multiple environments like DEV, QA, UAT, and PROD. This ensures that all environments are set up the same way, minimizing configuration drift and reducing the chances of errors due to different setups.

**-> CRUD Operations (Create, Read, Update, Delete):**

Terraform allows you to easily manage infrastructure by enabling you to create, read, update, and delete resources. It's like having a complete toolkit to interact with your infrastructure.

**-> Inventory Management:**

By looking at your Terraform code, you can see exactly what resources and services are being used for a specific project. It gives you a clear view of your infrastructure setup at any time, like an inventory list of what’s deployed.

**-> Cost Optimization:**

You can create infrastructure only when you need it and delete it when you no longer do. This helps you save costs by not keeping unused resources running, which is especially useful for cloud services.

**-> Dependency Management:**

Terraform automatically handles dependencies between resources. It knows which resources need to be created or updated first before others, ensuring that your infrastructure is built in the correct order.

**-> Modules (Code Reusability):**

Terraform lets you organize and reuse parts of your code as "modules." These modules act like building blocks that can be used in different parts of your infrastructure, making your code cleaner, more maintainable, and easier to manage.

**-> Declarative Language:**

Terraform uses a declarative language.

In short, Terraform allows you to automate and manage your infrastructure in a consistent, efficient, and cost-effective way by using code that is easy to version, reuse, and maintain.

* **Variables and data types (String, number, boolen, list and map)**
* **Conditions:**

**Expression ? “true statements” : “false statement”**

* **Loops ( Count and count index ):**

**For Loop (for each)**

**Dynamic Loop (Dynamic Block)**

* **Functions**
* **Locals (special type of variable)**
* **Data Sources**
* **Outputs – To print the output on to terminal**
* **State and remote state**
* **Tfvars (Multi-Env Like Dev, Prod,)**
* **Modules (Modules help centralize the code and allow us to reuse the same logic multiple times for different tasks or environments.)**
* **VPC, Subnets, Internet Gateways and route tables.**
* **NAT Gateway and VPC Peering**

**Creation of resources:**

**Provider**

**EC2**

**Security Group**

**Key Pair**

**Route53**

**Tags**

**VPC,**

**Subnets,**

**Internet Gateways**

**Route tables.**

**NAT Gateway**

**VPC Peering**