Infrastructure as Code (IaC) is a key practice in DevOps that involves managing and provisioning computing infrastructure through machine-readable definition files, rather than through physical hardware configuration or interactive configuration tools. IaC enables automation and simplifies the management of infrastructure, making it a cornerstone of modern DevOps practices.  
With Simple words – this is practice to have our infrastructure build and explaned with codes.

**Key Concepts of IaC:**

1. **Automation**:
   * Infrastructure is automatically provisioned and managed using scripts and definition files, reducing the need for manual configuration.
2. **Consistency**:
   * By using code to define infrastructure, environments can be consistently reproduced, ensuring that development, testing, and production environments are identical.
3. **Version Control**:
   * Infrastructure code can be versioned just like application code. This enables tracking of changes, rollback to previous versions, and collaboration among team members.
4. **Scalability**:
   * Infrastructure can be scaled up or down easily by modifying the code and reapplying it, making it adaptable to changing demands.
5. **Documentation**:
   * IaC files serve as documentation of the infrastructure, providing a clear and up-to-date view of the infrastructure setup.

**Examples of IaC Tools:**

1. **Terraform**:
   * Developed by HashiCorp, Terraform is an open-source tool that allows users to define and provision data center infrastructure using a declarative configuration language.

 Pros:

* Cloud-agnostic, supports multiple providers
* Strong community support and extensive documentation

 Cons:

* State management can be complex
* Learning curve for HCL (HashiCorp Configuration Language)
  + Example:

provider "aws" {

region = "us-west-2"

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

}

1. **AWS CloudFormation**:
   * A service by Amazon Web Services that provides a common language for describing and provisioning all the infrastructure resources in a cloud environment.
   * Example:

{

"Resources": {

"MyEC2Instance": {

"Type": "AWS::EC2::Instance",

"Properties": {

"ImageId": "ami-0c55b159cbfafe1f0",

"InstanceType": "t2.micro"

}

}

}

}

 Pros:

* Integrated with AWS services
* Supports a wide range of AWS resources

 Cons:

* Limited to AWS
* JSON/YAML syntax can be verbose and complex

1. **AWS CDK (Cloud Development Kit):**

* A framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation.
* Example:

from aws\_cdk import (

aws\_s3 as s3,

core

)

class S3SearchScriptStack(core.Stack):

def \_\_init\_\_(self, scope: core.Construct, id: str, \*\*kwargs) -> None:

super().\_\_init\_\_(scope, id, \*\*kwargs)

# Create an S3 bucket

self.bucket = s3.Bucket(self,

"SearchBucket",

versioned=True,

removal\_policy=core.RemovalPolicy.DESTROY # Not for production

)

 Pros:

* Allows using familiar programming languages (TypeScript, Python, Java, etc.)
* High-level constructs simplify resource definitions

 Cons:

* Requires knowledge of programming
* Can be more complex for simple tasks compared to declarative options