

Infrastructure as Code

Cloud Infrastructure Engineering

**Nanyang Technological University
& Skills Union - 2022/2023**

Course Content

- Quick Check-In
- Dive into what Infrastructure as Code is
- Dive into what AWS CDK is
- Explore the structure of AWS CDK project
- Explore usage of the AWS Construct Library to define AWS resources using code

Time	What	How or Why
7:15pm - 7:25pm	Part 1 - Presentation	Introduction to IaC
7:25pm - 7:45pm	Part 2 - Presentation	Installation steps
7:45pm - 8:10pm	Part 3 - Presentation	AWS CDK
8:10pm - 8:20pm	Break	
8:20pm - 8:50pm	Part 4 - Activity	AWS CDK
8:50pm - 10:00pm	Summary & Assignments	

Recap

- Operational Excellence Principles
 - Perform Operations as Code
 - Make Frequent, Small, Reversible Changes
 - Refine Operations Procedures Frequently
 - Anticipate Failure
 - Learn From All Operational Failures
- Best Practices
 - Organization
 - Prepare
 - Operate
 - Evolve

Self Study Check-In



Q1) What is IaC?



Q2) What are some examples of IaC?



- 3) What is the primary goal of Infrastructure as Code?
- a) To eliminate the need for network administrators.
 - b) To automate the deployment and management of infrastructure.
 - c) To replace physical infrastructure with virtual machines.
 - d) To reduce the complexity of network architecture.

Overview



Overview

A fundamental principle of DevOps is to **treat infrastructure the same way developers treat code.**

Application code has a defined format and syntax.

If the code is not written according to the rules of the programming language, applications cannot be created.

Overview

Code is stored in a **version management or source control system** that logs a history of code development, changes, and bug fixes.

When code is compiled or built into applications, we expect a consistent application to be created, and the build is repeatable and reliable.

Infrastructure as Code










IaC

Practicing infrastructure as code means applying the same rigor of application code development to infrastructure provisioning.

All configurations should be defined in a declarative way and stored in a source control system.

IaC

CONTINUOUS IT STACK

1		INFRASTRUCTURE as code	In the world of IT operations, infrastructure as code is responsible for on-boarding, provisioning, and licensing of infrastructure and services necessary to deploy and operate the deployment pipeline. This includes containers.	TOOLS: 
2		CONFIGURATION as code	Once infrastructure is prepped, it must be configured. Configuration as Code uses declarative or imperative methods to automatically configure services in the deployment pipeline.	TOOLS: 
3		PIPELINE as code	The pipeline drives the deployment of an application from start to finish. This layer of the stack is what defines per-application pipelines and enables continuous operation of a comprehensive data path that secures and scales an application.	TOOLS: 
4		OPERATIONS as code	Operations as code is concerned with post-deployment operations. This includes capabilities like auto-scaling, monitoring and alerting, and resource discovery. These are repeatable workflows (processes) that comprise a variety of operational tasks for which automation makes sense.	TOOLS:

IaC



AWS CloudFormation



Pulumi



Terraform



aws
**Cloud
Development
Kit**

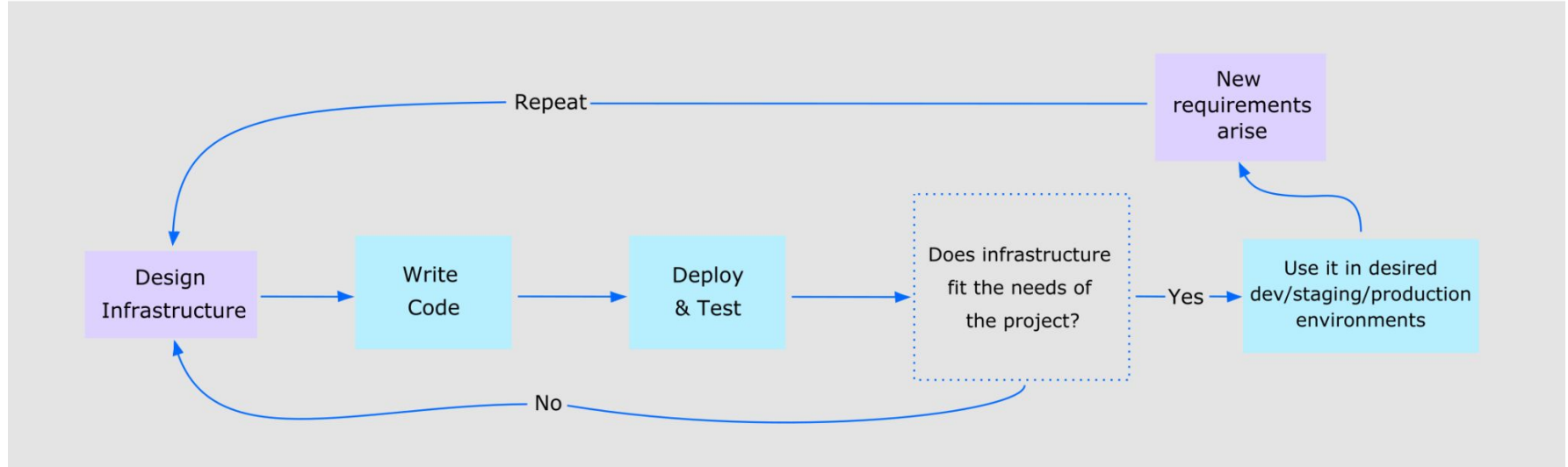


Google
Cloud Deployment
Manager

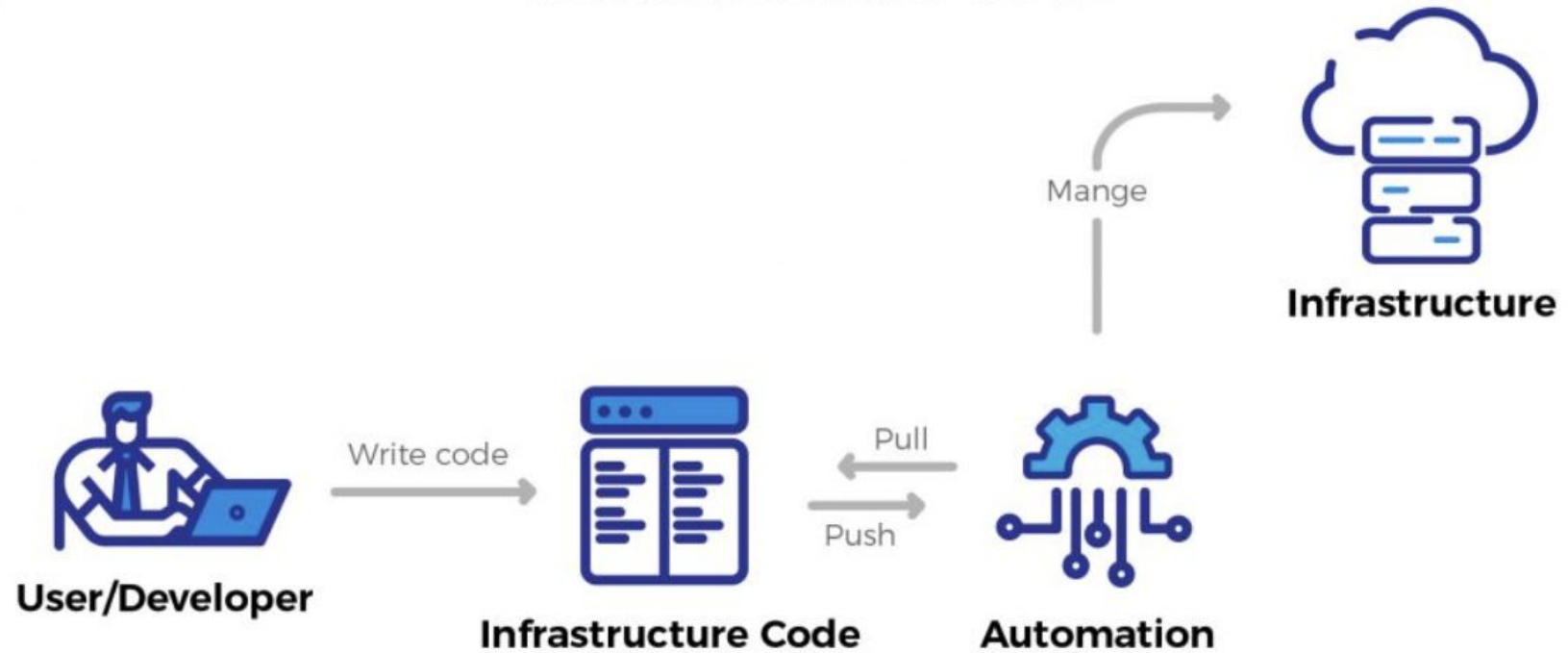


**AZURE
RESOURCEMANAGER**

IaC Workflow



IaC



laC

Infrastructure was traditionally provisioned using a combination of scripts and manual processes.

Sometimes these scripts were stored in version control systems or documented step by step in text files or run-books.

Often the person writing the run books is not the same person executing these scripts or following through the run-books.

IaC

If these scripts or runbooks are not updated frequently, they can **potentially become a show-stopper in deployments.**

This results in the creation of new environments **not always being repeatable, reliable, or consistent.**

Activity - Let's Install

- Step 1: Install AWS CLI using this link:
<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>
- Step 2: Add AWS Credential to your machine:
<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html>

AWS CDK



AWS CDK

The AWS Cloud Development Kit (AWS CDK) lets you define your cloud infrastructure as code in one of its supported programming languages.

It is intended for moderately to highly experienced AWS users.

Advantages of AWS CDK

- Build with high-level constructs that give appropriate, secure defaults for your *AWS* resources, allowing you to define more infrastructure with less code.
- Put your infrastructure, application code, and configuration in one location to ensure that you have a complete, cloud-deployable solution at each milestone.
- Use software engineering methods such as code reviews, unit tests, and source control to strengthen your infrastructure.
- You may easily share infrastructure design patterns inside your organization or with the general world.

AWS CDK Benefits

- Easier cloud onboarding
- Faster development process
- Customizable and shareable
- No context switching

AWS CDK



**Cloud
Development
Kit**

AWS CDK

The AWS CDK is designed around a handful of important concepts. We will introduce a few of these here briefly.

An AWS CDK app is **an application written in TypeScript, JavaScript, Python, Java, C# or Go** that uses the AWS CDK to define AWS infrastructure. An **app defines one or more stacks**.

Stacks (equivalent to AWS CloudFormation stacks) contain constructs.

Each construct defines one or more concrete AWS resources, such as Amazon S3 buckets, Lambda functions, or Amazon DynamoDB tables.

AWS CDK



Use preconfigured application components

Download preconfigured components from a package manager or artifact repository



Model your application

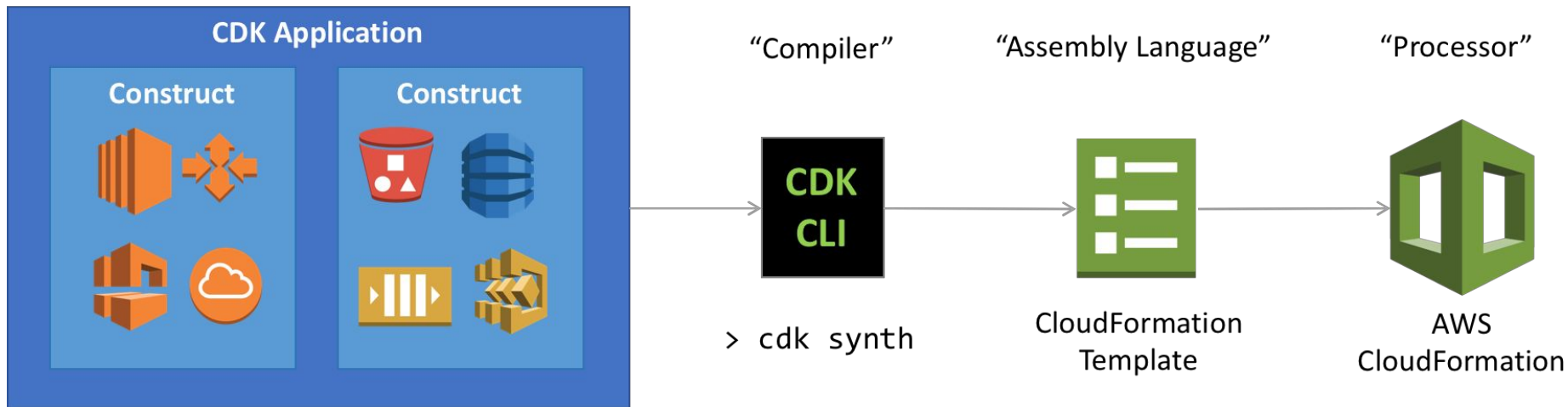
Model your application logic and infrastructure in a programming language



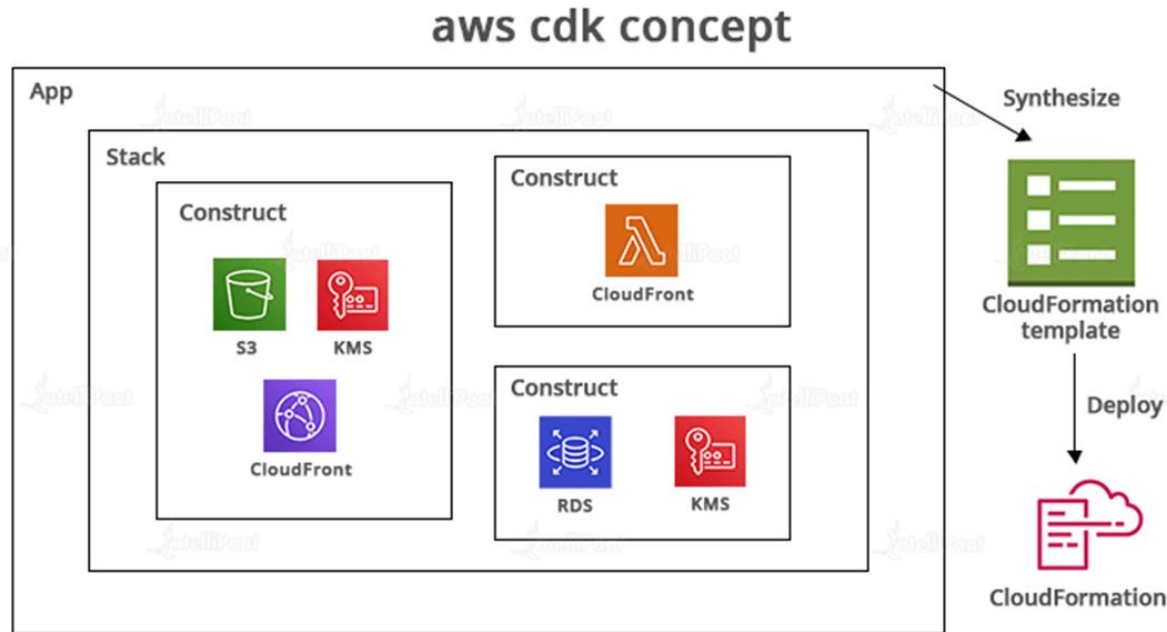
Provision your application with AWS CloudFormation

Provision your application code and supporting infrastructure with AWS CloudFormation

AWS CDK Concept



AWS CDK Concept



There are three core concepts of AWS CDK:

App

This is the base of your construct tree and combines all stacks and constructs into a single application that can then be deployed on AWS Cloud.

Stacks

Stacks are basic deployable components in AWS CDK. A project can contain one or more stacks. By consuming resources from another stack, a stack can share values. AWS CDK leverages the CloudFormation import value to pass around different CloudFormation stacks behind the scenes.

Constructs

AWS CDK stacks, like CloudFormation stacks, comprise constructs that describe one or more AWS resources. Constructs include everything CloudFormation requires to build a reusable cloud component.

AWS CDK

Constructs (and also stacks and apps) are represented as classes (types) in your programming language of choice.

You **instantiate constructs within a stack to declare them to AWS**, and connect them to each other using well-defined interfaces.

The AWS CDK includes the **CDK Toolkit** (also called the CLI), a **command line tool** for working with your AWS CDK apps and stacks. Among other functions, the Toolkit provides the ability to do the following:

- Convert one or more AWS CDK stacks to AWS CloudFormation templates and related assets (a process called synthesis)
- Deploy your stacks to an AWS account

AWS CDK

The AWS CDK includes a **library of AWS constructs called the AWS Construct Library**, organized into various modules.

The library contains constructs for each AWS service.

The main CDK package is called **aws-cdk-lib**, and it contains the majority of the AWS Construct Library.

It also contains base classes like Stack and App that are used in most CDK applications.

More information: <https://docs.aws.amazon.com/cdk/v2/guide/home.html>

Using AWS CDK



Activity - Installing AWS CDK

Run -

Install npm and node: [Download | Node.js \(nodejs.org\)](https://nodejs.org/en/download/)

```
node -v
```

```
npm -v
```

```
npm install -g aws-cdk
```

```
cdk --version
```

Activity - Ensuring aws is configured

Run -

```
aws sts get-caller-identity
```

output:

```
{  
  "UserId": "AIDATXF4JQPH4LSMA5XE7",  
  "Account": "255945442255",  
  "Arn": "arn:aws:iam::255945442255:user/luqman"  
}
```

Activity - Ensuring aws is configured

Run -

If you have not configured aws

aws configure

Activity - Exploring AWS CDK

1. Create the folder using this command

```
mkdir hello-cdk  
cd hello-cdk
```

2. Create the app using this command

```
cdk init app --language typescript
```

3. Build the app

```
npm run build
```

Activity - Exploring AWS CDK

In `lib/hello-cdk-stack.ts`:

```
import * as cdk from 'aws-cdk-lib';
import { aws_s3 as s3 } from 'aws-cdk-lib';

export class HelloCdkStack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}
```

Activity - Deploy AWS CDK

Run -

```
cdk bootstrap
```

```
cdk deploy
```

Activity - Modify AWS CDK

Update `lib/hello-cdk-stack.ts`.

```
new s3.Bucket(this, 'MyFirstBucket', {  
  versioned: true,  
  removalPolicy: cdk.RemovalPolicy.DESTROY,  
  autoDeleteObjects: true  
});
```



Activity - Modify AWS CDK

Run -

```
cdk diff
```

What do you see?

Activity - Deploy AWS CDK again

Run -

```
cdk deploy
```

Activity - Deleting AWS CDK

Run -

```
cdk destroy
```

Terraform



Terraform

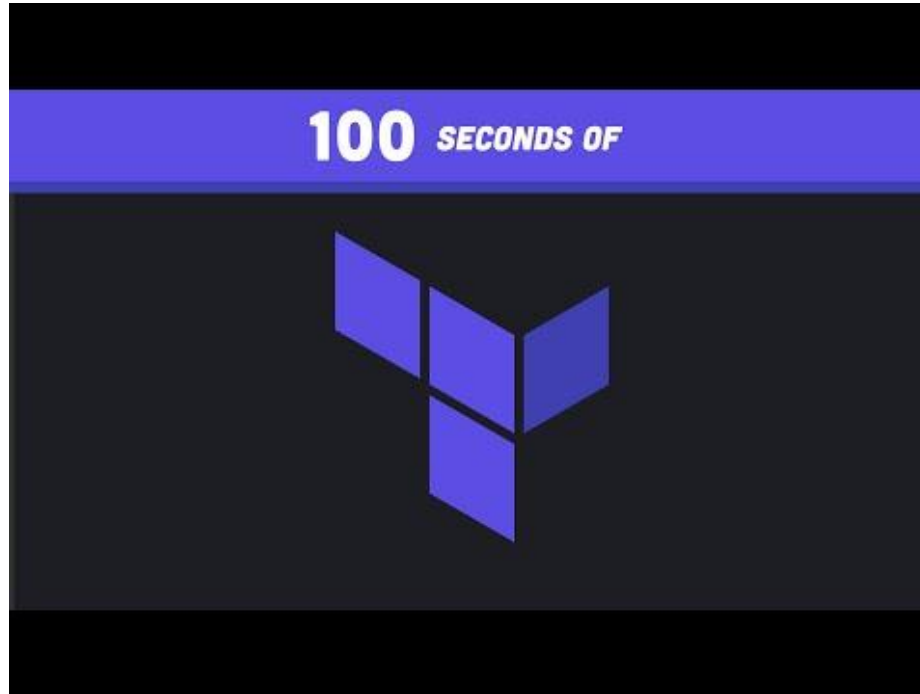
Infrastructure as code (IaC) tools allow you to manage infrastructure with configuration files rather than through a graphical user interface. IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.

Terraform is HashiCorp's infrastructure as code tool. It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle.

Advantages Using Terraform

- Terraform can manage infrastructure on multiple cloud platforms.
- The human-readable configuration language helps you write infrastructure code quickly.
- Terraform's state allows you to track resource changes throughout your deployments.
- You can commit your configurations to version control to safely collaborate on infrastructure.

Let's Explore IaC



We will explore more about Terraform in

2.16 Infrastructure as Code - 2

2.16 Infrastructure as Code - 3

IaC - Terraform

```
ec2.tf > ...  
1  resource "aws_instance" "example" {  
2      ami          = "ami-0cff7528ff583bf9a"  
3      instance_type = "t3.micro"  
4  
5      tags = {  
6          Name = "Example VM"  
7      }  
8  }  
9
```

IaC - Terraform

```
provider "aws" {  
  region      = var.region  
  access_key  = var.access_key  
  secret_key  = var.secret_key  
  default_tags{  
    tags = {  
      Source = "terraform-module-16"  
      Pipeline = "terraform-mod-16-CI"  
    }  
  }  
}
```

IaC - CloudFormation

```
Resources:
  MyLambdaFunction:
    Type: "AWS::Lambda::Function"
    Properties:
      Code:
        ZipFile: |
          def lambda_handler(event, context):
            print("Hello, World!")
            return "Hello, World!"
      Handler: index.lambda_handler
      Role: !GetAtt MyLambdaRole.Arn
      Runtime: python3.7
      Timeout: 3

  MyLambdaRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: "Allow"
            Principal:
              Service: "lambda.amazonaws.com"
            Action: "sts:AssumeRole"
```

Activity

Learner:

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.
- Check the AWS account after learner clean up.

What's Next?

