1. Basic VPC Setup Template:

* AWSTemplateFormatVersion: 2010-09-09
* *# This CloudFormation template deploys a basic VPC / Network.  Specifically:*
* *# It deploys a VPC with 4 subnets (2 public, 2 private) across 2 Availability Zones*
* *# A NAT Gateway is used to provide egress for private subnets.*
* *# The VPC and subnets are exported for use by other stacks.*
* Resources:
* *# First, a VPC:*
* VPC:
* Type: AWS::EC2::VPC
* Properties:
* CidrBlock: 10.1.0.0/16
* EnableDnsSupport: true
* EnableDnsHostnames: true
* Tags:
* - Key: Name
* Value:  !Join ['', [!Ref "AWS::StackName", "-VPC" ]]
* *# Our VPC will need internet access:*
* InternetGateway:
* Type: AWS::EC2::InternetGateway
* DependsOn: VPC
* AttachGateway:
* Type: AWS::EC2::VPCGatewayAttachment
* *# Notice how you can't attach an IGW to a VPC unless both are created:*
* Properties:
* VpcId: !Ref VPC
* InternetGatewayId: !Ref InternetGateway
* *# Now some subnets, 2 public and 2 private:*
* PublicSubnetA:
* Type: AWS::EC2::Subnet
* Properties:
* VpcId: !Ref VPC
* CidrBlock: 10.1.10.0/24
* AvailabilityZone: !Select [ 0, !GetAZs ]    *# Get the first AZ in the list*
* Tags:
* - Key: Name
* Value: !Sub ${AWS::StackName}-Public-A
* PublicSubnetB:
* Type: AWS::EC2::Subnet
* Properties:
* VpcId: !Ref VPC
* CidrBlock: 10.1.20.0/24
* AvailabilityZone: !Select [ 1, !GetAZs ]    *# Get the second AZ in the list*
* Tags:
* - Key: Name
* Value: !Sub ${AWS::StackName}-Public-B
* PrivateSubnetA:
* Type: AWS::EC2::Subnet
* Properties:
* VpcId: !Ref VPC
* CidrBlock: 10.1.50.0/24
* AvailabilityZone: !Select [ 0, !GetAZs ]    *# Get the first AZ in the list*
* Tags:
* - Key: Name
* Value: !Sub ${AWS::StackName}-Private-A
* PrivateSubnetB:
* Type: AWS::EC2::Subnet
* Properties:
* VpcId: !Ref VPC
* CidrBlock: 10.1.60.0/24
* AvailabilityZone: !Select [ 1, !GetAZs ]    *# Get the second AZ in the list*
* Tags:
* - Key: Name
* Value: !Sub ${AWS::StackName}-Private-B
* *# Some route tables for our subnets:*
* PublicRouteTable:
* Type: AWS::EC2::RouteTable
* Properties:
* VpcId: !Ref VPC
* Tags:
* - Key: Name
* Value: Public
* PublicRoute1:   *# Public route table has direct routing to IGW:*
* Type: AWS::EC2::Route
* DependsOn: AttachGateway
* Properties:
* RouteTableId: !Ref PublicRouteTable
* DestinationCidrBlock: 0.0.0.0/0
* GatewayId: !Ref InternetGateway
* *# Here is a private route table:*
* PrivateRouteTable:
* Type: AWS::EC2::RouteTable
* Properties:
* VpcId: !Ref VPC
* Tags:
* - Key: Name
* Value: Private
* PrivateRoute1:            *# Private route table can access web via NAT (created below)*
* Type: AWS::EC2::Route
* Properties:
* RouteTableId: !Ref PrivateRouteTable
* DestinationCidrBlock: 0.0.0.0/0
* *# Route traffic through the NAT Gateway:*
* NatGatewayId: !Ref NATGateway
* *# Attach the public subnets to public route tables,*
* *# and attach the private subnets to private route tables:*
* PublicSubnetARouteTableAssociation:
* Type: AWS::EC2::SubnetRouteTableAssociation
* Properties:
* SubnetId: !Ref PublicSubnetA
* RouteTableId: !Ref PublicRouteTable
* PublicSubnetBRouteTableAssociation:
* Type: AWS::EC2::SubnetRouteTableAssociation
* Properties:
* SubnetId: !Ref PublicSubnetB
* RouteTableId: !Ref PublicRouteTable
* PrivateSubnetARouteTableAssociation:
* Type: AWS::EC2::SubnetRouteTableAssociation
* Properties:
* SubnetId: !Ref PrivateSubnetA
* RouteTableId: !Ref PrivateRouteTable
* PrivateSubnetBRouteTableAssociation:
* Type: AWS::EC2::SubnetRouteTableAssociation
* Properties:
* SubnetId: !Ref PrivateSubnetB
* RouteTableId: !Ref PrivateRouteTable
* *# A NAT Gateway:*
* NATGateway:
* Type: AWS::EC2::NatGateway
* Properties:
* AllocationId: !GetAtt ElasticIPAddress.AllocationId
* SubnetId: !Ref PublicSubnetA
* Tags:
* - Key: Name
* Value: !Sub NAT-${AWS::StackName}
* ElasticIPAddress:
* Type: AWS::EC2::EIP
* Properties:
* Domain: VPC

1. **EC2 instance with Apache web server**:

* AWSTemplateFormatVersion: 2010-09-09
* *# This CloudFormation template will setup an EC2 instance with apache server installed on it. The default region is us-east-1.*
* Resources:
* EC2ApacheInstance:
* Type: AWS::EC2::Instance
* Properties:
* ImageId: ami-0ae8f15ae66fe8cda
* InstanceType: t2.micro
* Tags:
* - Key: Env
* Value: Dev
* - Key: Name
* Value: Apache Web Server
* UserData: !Base64
* Fn::Sub: |
* #!/bin/sh
* yum update -y
* yum install httpd.x86\_64 -y
* systemctl start httpd.service
* systemctl enable httpd.service
* echo "Hello World" > /var/www/html/index.html
* SecurityGroups:
* - !Ref ApacheSecurityGroup
* KeyName: sample-apache
* ApacheSecurityGroup:
* Type: AWS::EC2::SecurityGroup
* Properties:
* GroupName: EC2 Group
* GroupDescription: EC2 Group
* SecurityGroupIngress:
* - IpProtocol: tcp
* CidrIp: 0.0.0.0/0
* FromPort: 22
* ToPort: 22
* - IpProtocol: tcp
* CidrIp: 0.0.0.0/0
* FromPort: 80
* ToPort: 80
* - IpProtocol: tcp
* CidrIp: 0.0.0.0/0
* FromPort: 443
* ToPort: 443

1. Simple S3 Bucket Setup with Encryption and Versioning:

* AWSTemplateFormatVersion: '2010-09-09'
* Parameters:
* BucketName:
* Type: String
* Description: Specify the name for the S3 Bucket.
* AllowedPattern: "^([a-z0-9]{1}[a-z0-9-]{1,61}[a-z0-9]{1})$"
* ConstraintDescription: "Bucket names must be between 3 and 63 characters, start and end with a letter or number."
* Encryption:
* Type: String
* Description: Enable default encryption for the S3 bucket.
* AllowedValues:
* - "true"
* - "false"
* Default: "true"
* Versioning:
* Type: String
* Description: Enable or suspend versioning for objects in the S3 bucket.
* AllowedValues:
* - "Enabled"
* - "Suspended"
* Default: "Suspended"
* Resources:
* MyS3Bucket:
* Type: AWS::S3::Bucket
* Properties:
* BucketName: !Ref BucketName
* VersioningConfiguration:
* Status: !Ref Versioning
* BucketEncryption:
* !If
* - IsEncrypted
* - ServerSideEncryptionConfiguration:
* - ServerSideEncryptionByDefault:
* SSEAlgorithm: AES256
* - !Ref "AWS::NoValue"
* Conditions:
* IsEncrypted: !Equals [!Ref Encryption, "true"]
* Outputs:
* BucketURL:
* Description: "URL for accessing the S3 bucket"
* Value: !Join ['', ["https://", !GetAtt MyS3Bucket.DomainName]]
* BucketARN:
* Description: "ARN of the S3 bucket"
* Value: !GetAtt MyS3Bucket.Arn

1. Amazon Relational Database Service (RDS) Setup:

* AWSTemplateFormatVersion: '2010-09-09'
* Parameters:
* myDbName:
* Type: String
* Description: "The name of the database."
* myDbUser:
* Type: String
* Description: "The master username for the database."
* myDbPass:
* Type: String
* NoEcho: true
* MinLength: 8
* MaxLength: 41
* Description: "The master user password for the database. Must be between 8 and 41 characters."
* Resources:
* SQLDemoDbInstance:
* Type: AWS::RDS::DBInstance
* Properties:
* DBName: !Ref myDbName
* MasterUsername: !Ref myDbUser
* MasterUserPassword: !Ref myDbPass
* Engine: MySQL
* DBInstanceClass: db.t3.micro
* StorageType: gp2
* PubliclyAccessible: true
* AllocatedStorage: "20"
* DBInstanceIdentifier: !Sub "SQLDemoDbInstance-${AWS::Region}"
* MultiAZ: false
* BackupRetentionPeriod: 7
* AvailabilityZone: !Select [1, !GetAZs ""]
* Outputs:
* DBInstanceEndpoint:
* Description: "RDS MySQL Database Endpoint"
* Value: !GetAtt SQLDemoDbInstance.Endpoint.Address

1. Deploy a Serverless static website using AWS CloudFront and S3 bucket:

* AWSTemplateFormatVersion: 2010-09-09
* Description: Static website hosting with S3 and CloudFront with a custom domain.
* Parameters:
* Cert:
* Description: SSL Certificate ARN
* Type: String
* HostedZoneResourceID:
* Description: Hosted Zone ID
* Type: String
* DomainName:
* Description: Website Domain Name
* Type: String
* ErrorPagePath:
* Description: Directory error path
* Type: String
* Default: /error.html
* IndexDocument:
* Description: Directory index path
* Type: String
* Default: /index.html
* Resources:
* S3Bucket:
* Type: "AWS::S3::Bucket"
* Properties:
* BucketName: !Sub ${DomainName}-cloudfront
* CloudFrontOriginAccessIdentity:
* Type: "AWS::CloudFront::CloudFrontOriginAccessIdentity"
* Properties:
* CloudFrontOriginAccessIdentityConfig:
* Comment: !Ref S3Bucket
* ReadPolicy:
* Type: "AWS::S3::BucketPolicy"
* Properties:
* Bucket: !Ref S3Bucket
* PolicyDocument:
* Statement:
* - Action: "s3:GetObject"
* Effect: Allow
* Resource:
* - !Sub "${S3Bucket.Arn}"
* - !Sub "${S3Bucket.Arn}/\*"
* Principal:
* CanonicalUser: !GetAtt CloudFrontOriginAccessIdentity.S3CanonicalUserId
* CloudFrontDistribution:
* Type: "AWS::CloudFront::Distribution"
* Properties:
* DistributionConfig:
* Aliases:
* - !Ref DomainName
* ViewerCertificate:
* AcmCertificateArn: !Ref Cert
* SslSupportMethod: sni-only
* CustomErrorResponses:
* - ErrorCode: 403
* ResponseCode: 404
* ResponsePagePath: !Ref ErrorPagePath
* DefaultCacheBehavior:
* AllowedMethods:
* - GET
* - HEAD
* - OPTIONS
* CachedMethods:
* - GET
* - HEAD
* - OPTIONS
* Compress: true
* DefaultTTL: 3600
* ForwardedValues:
* Cookies:
* Forward: none
* QueryString: false
* MaxTTL: 86400
* MinTTL: 120
* TargetOriginId: s3origin
* ViewerProtocolPolicy: redirect-to-https
* DefaultRootObject: "index.html"
* Enabled: true
* HttpVersion: http2
* Origins:
* - DomainName: !GetAtt S3Bucket.DomainName
* Id: s3origin
* S3OriginConfig:
* OriginAccessIdentity: !Sub >-
* origin-access-identity/cloudfront/${CloudFrontOriginAccessIdentity}
* PriceClass: PriceClass\_All
* User:
* Type: "AWS::IAM::User"
* Properties:
* Policies:
* - PolicyName: !Sub "publish-to-${S3Bucket}"
* PolicyDocument:
* Statement:
* - Action: "s3:\*"
* Effect: Allow
* Resource:
* - !Sub "${S3Bucket.Arn}"
* - !Sub "${S3Bucket.Arn}/\*"
* DNSRecord:
* Type: AWS::Route53::RecordSet
* Properties:
* HostedZoneId: !Ref HostedZoneResourceID
* Comment: DNS name for cloud front
* Name: !Ref DomainName
* Type: A
* AliasTarget:
* HostedZoneId: Z2FDTNDATAQYW2
* DNSName: !GetAtt CloudFrontDistribution.DomainName
* DependsOn: CloudFrontDistribution
* Outputs:
* BucketName:
* Description: S3 Bucket Name
* Value: !Ref S3Bucket
* URL:
* Description: Website URL
* Value: !Ref DNSRecord

1. SNS Topic with Email Subscription:

* AWSTemplateFormatVersion: '2010-09-09'
* Description: SNS Topic with an email subscription for notifications.
* Parameters:
* EmailAddress:
* Type: String
* Description: Email address to receive notifications
* Resources:
* MySNSTopic:
* Type: AWS::SNS::Topic
* Properties:
* DisplayName: "MyNotificationTopic"
* MySubscription:
* Type: AWS::SNS::Subscription
* Properties:
* TopicArn: !Ref MySNSTopic
* Protocol: email
* Endpoint: !Ref EmailAddress

1. SQS Queue with Dead Letter Queue:

* AWSTemplateFormatVersion: '2010-09-09'
* Description: SQS Queue with a Dead Letter Queue (DLQ).
* Resources:
* MyQueue:
* Type: AWS::SQS::Queue
* Properties:
* QueueName: MyQueue
* RedrivePolicy:
* deadLetterTargetArn: !GetAtt MyDeadLetterQueue.Arn
* maxReceiveCount: 5
* MyDeadLetterQueue:
* Type: AWS::SQS::Queue
* Properties:
* QueueName: MyDeadLetterQueue

1. Amazon Kinesis DataStream Setup:

* AWSTemplateFormatVersion: '2010-09-09'
* Description: CloudFormation template to create an Amazon Kinesis Data Stream
* Resources:
* KinesisDataStream:
* Type: AWS::Kinesis::Stream
* Properties:
* Name: MyKinesisDataStream
* ShardCount: 1  # Number of shards, adjust as per requirements
* KinesisStreamIAMRole:
* Type: AWS::IAM::Role
* Properties:
* RoleName: KinesisStreamRole
* AssumeRolePolicyDocument:
* Version: '2012-10-17'
* Statement:
* - Effect: Allow
* Principal:
* Service:
* - kinesis.amazonaws.com
* Action: sts:AssumeRole
* Policies:
* - PolicyName: KinesisStreamPolicy
* PolicyDocument:
* Version: '2012-10-17'
* Statement:
* - Effect: Allow
* Action:
* - kinesis:DescribeStream
* - kinesis:GetRecords
* - kinesis:GetShardIterator
* - kinesis:PutRecord
* - kinesis:PutRecords
* Resource:
* !GetAtt KinesisDataStream.Arn
* Outputs:
* KinesisStreamName:
* Description: Name of the Kinesis Data Stream
* Value: !Ref KinesisDataStream
* KinesisStreamArn:
* Description: ARN of the Kinesis Data Stream
* Value: !GetAtt KinesisDataStream.Arn

1. Deploying a Simple AWS Lambda Function:

* AWSTemplateFormatVersion: '2010-09-09'
* Description: CloudFormation template to create a simple AWS Lambda function that returns "Hello, World!".
* Resources:
* MyLambdaFunction:
* Type: AWS::Lambda::Function
* Properties:
* FunctionName: SimpleHelloWorldFunction
* Handler: index.handler
* Role: !GetAtt LambdaExecutionRole.Arn
* Runtime: nodejs18.x
* Code:
* ZipFile: |
* exports.handler = async (event) => {
* const response = {
* statusCode: 200,
* body: JSON.stringify('Hello, World!'),
* };
* return response;
* };
* LambdaExecutionRole:
* Type: AWS::IAM::Role
* Properties:
* AssumeRolePolicyDocument:
* Version: '2012-10-17'
* Statement:
* - Effect: Allow
* Principal:
* Service: lambda.amazonaws.com
* Action: 'sts:AssumeRole'
* Policies:
* - PolicyName: LambdaBasicExecution
* PolicyDocument:
* Version: '2012-10-17'
* Statement:
* - Effect: Allow
* Action:
* - 'logs:CreateLogGroup'
* - 'logs:CreateLogStream'
* - 'logs:PutLogEvents'
* Resource: 'arn:aws:logs:\*:\*:\*'
* Outputs:
* LambdaFunctionName:
* Description: "The name of the Lambda function"
* Value: !Ref MyLambdaFunction
* LambdaFunctionARN:
* Description: "The ARN of the Lambda function"
* Value: !GetAtt MyLambdaFunction.Arn