AWS CDK in a Nutshell

Stephan Rauh

March 2022

1 Why?

- Infrastructure as Code is a great idea
- YAML files are not code (though Terraform tries hard)
- Wanted: variables, methods, splitting stuff into multiple files
- Wanted: refactoring
- Wanted:
  short, concise scripts

1 Getting to Speed Fast

```
export class CdkStack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);
    const blogBucket = createBlogBucket(this);
    const commentBucket = createCommentBucket(this);
    const readCommentsLambda = createLambdaReadingComments(this);
    commentBucket.grantRead(readCommentsLambda);
    const writeCommentsLambda = createLambdaWritingComments(this);
    commentBucket.grantReadWrite(writeCommentsLambda);
    const [apiKey, gatewayId] = createGateway(this, readCommentsLambda,
                                              writeCommentsLambda);
    const cf = createCloudfrontDistribution(this, blogBucket, gatewayId, apiKey);
    createLambda(this, '/../src/emergency-mode/activate.ts', cf);
  }
}
```

1 Bye bye Boilerplate!

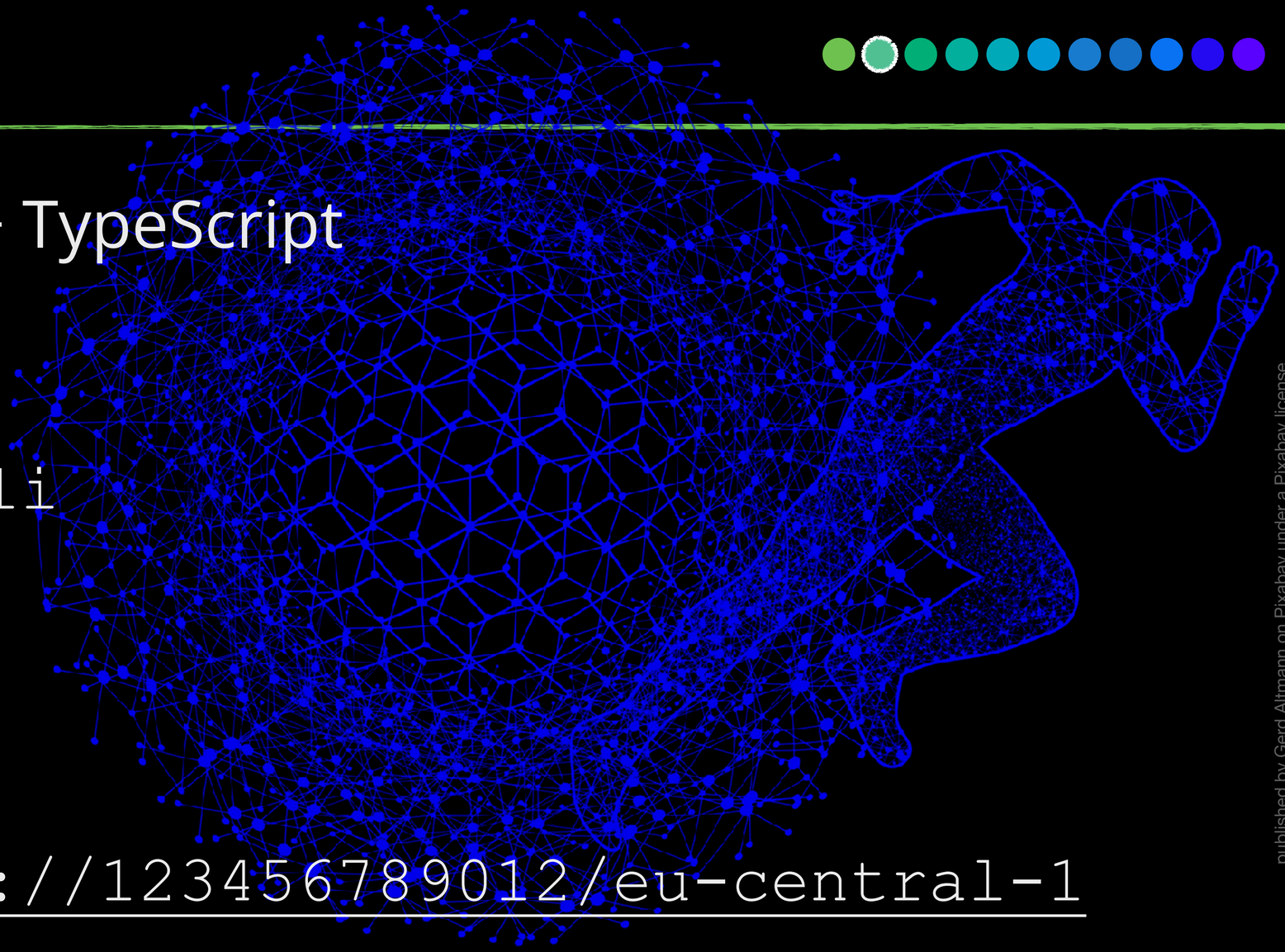
```
export class CdkStack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);
    const blogBucket = createBlogBucket(this);
    const commentBucket = createCommentBucket(this);
    const readCommentsLambda = createLambdaReadingComments(this);
    commentBucket.grantRead(readCommentsLambda);
    const writeCommentsLambda = createLambdaWritingComments(this);
    commentBucket.grantReadWrite(writeCommentsLambda);
    const [apiKey, gatewayId] = createGateway(this, readCommentsLambda,
                                              writeCommentsLambda);
    const cf = createCloudfrontDistribution(this, blogBucket, gatewayId, apiKey);
    createLambda(this, '/../src/emergency-mode/activate.ts', cf);
  }
}
```

1 Same Stack with Cloudformation

```
Resources:
  cdkblogbucketB729DF96:
    Type: AWS::S3::Bucket
    Properties:
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256
      BucketName: cdk-blog-bucket
      CorsConfiguration:
        CorsRules:
          - AllowedHeaders:
              - "*"
            AllowedMethods:
              - GET
              - POST
              - PUT
            AllowedOrigins:
              - http://localhost:4200
            MaxAge: 3600
      LifecycleConfiguration:
        Rules:
          - AbortIncompleteMultipartUpload:
              DaysAfterInitiation: 1
            Status: Enabled
            Transitions:
              - StorageClass: INTELLIGENT_TIERING
                TransitionInDays: 30
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
      Tags:
        - Key: aws-cdk:auto-delete-objects
          Value: "true"
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
    Metadata:
      aws:cdk:path: BlogStack/cdk-blog-bucket/Resource
  cdkblogbucketPolicy9A43866A:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket:
        Ref: cdkblogbucketB729DF96
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092
                  - Arn
            Resource:
              - Fn::GetAtt:
                  - cdkblogbucketB729DF96
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - cdkblogbucketB729DF96
                        - Arn
                    - /*
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - :iam::123456789012:root
            Resource:
              - Fn::GetAtt:
                  - cdkblogbucketB729DF96
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - cdkblogbucketB729DF96
                        - Arn
                    - /*
          - Action: s3:GetObject
            Effect: Allow
            Principal:
              CanonicalUser:
                Fn::GetAtt:
                  - cdkblogcdnOrigin1S3Origin7097BF81
                  - S3CanonicalUserId
            Resource:
              Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - cdkblogbucketB729DF96
                      - Arn
                  - /*
        Version: "2012-10-17"
```

- Plus 1000 lines
- Code snippet describes a single S3 bucket

2 First Steps

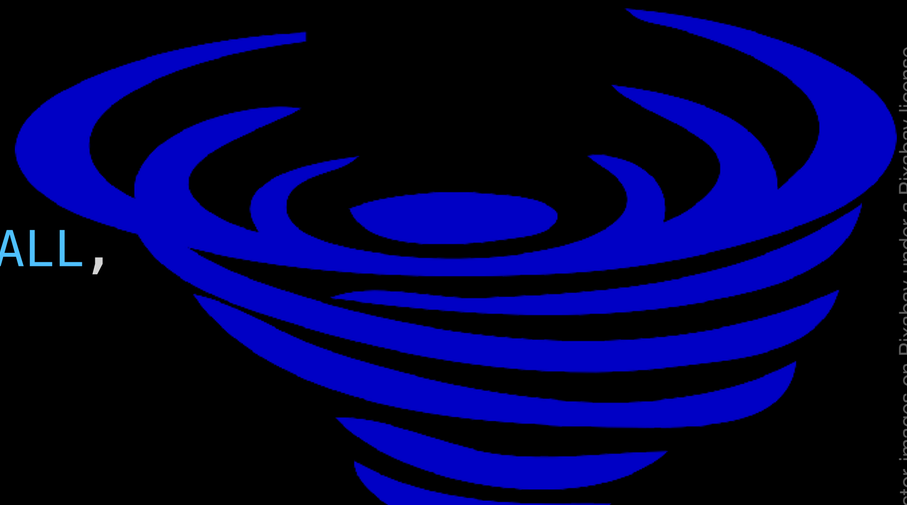


- Recommended: npm + TypeScript
  `brew install node`
- Install AWS CLI:
  `brew install awscli`
- Install CDK globally:
  `npm -g cdk`
- Bootstrap CDK
  `cdk bootstrap aws://123456789012/eu-central-1`
- Create demo project:
  `cdk init app --language=typescript`

3 Creating an S3 Bucket

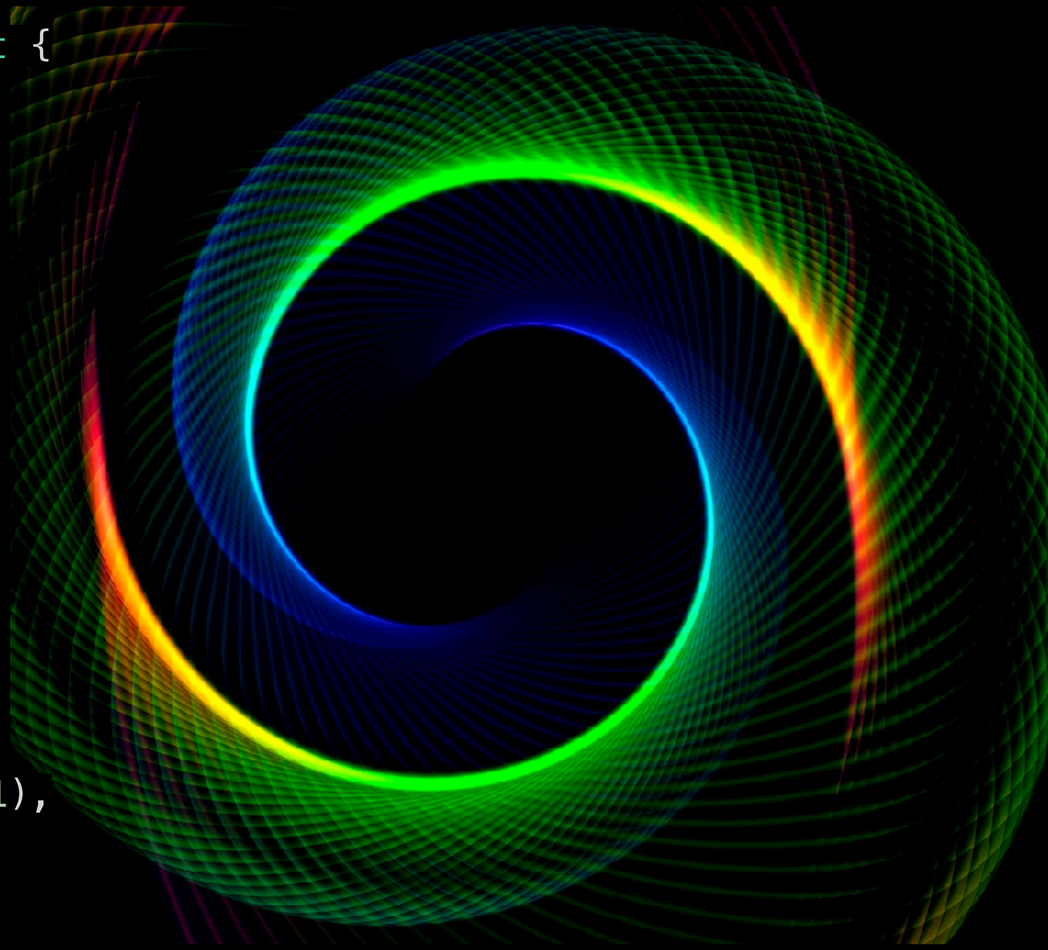
```
export class CdkStack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    new s3.Bucket(this, 'example-bucket', {
      bucketName: 'example-bucket',
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
      encryption: s3.BucketEncryption.S3_MANAGED,
    });
  }
}
```



3 Creating an S3 Bucket

```
function createBucket(name: string, stack: cdk.Stack): s3.Bucket {
  return new s3.Bucket(stack, name + '-bucket', {
    bucketName: name + '-bucket',
    blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
    removalPolicy: cdk.RemovalPolicy.DESTROY,
    autoDeleteObjects: true,
    publicReadAccess: false,
    encryption: s3.BucketEncryption.S3_MANAGED,
    cors: [
      {
        allowedMethods: [s3.HttpMethods.GET],
        allowedOrigins: ['http://localhost:4200'],
        allowedHeaders: ['*'],
        maxAge: cdk.Duration.hours(1).toSeconds(),
      },
    ],
    lifecycleRules: [
      {
        abortIncompleteMultipartUploadAfter: cdk.Duration.days(1),
        transitions: [
          {
            storageClass: s3.StorageClass.INTELLIGENT_TIERING,
            transitionAfter: cdk.Duration.days(30),
          },
        ],
      },
    ],
  });
}
```



4 Creating an AWS Lambda

```
function createNodeJSFunction(stack: cdk.Stack,
                              name: string,
                              source: string,
                              description: string): NodejsFunction {
  const myFunction = new NodejsFunction(stack, name, {
    functionName: name,
    description,
    memorySize: 1024,
    timeout: cdk.Duration.seconds(5),
    runtime: lambda.Runtime.NODEJS_14_X,
    entry: path.join(__dirname, source),
    bundling: {
      minify: true,
    },
  });
  return myFunction;
}
```

4 Link S3 Bucket with an AWS Lambda

```
const bucket = createCommentBucket(this);
const reader = createNodeJSFunction(stack,
                'readerLambda',
                `/../src/read/index.ts`,
                'Read and process the bucket');
bucket.grantRead(reader);
```

4 IAM without the Pain

```
bucket.grantRead(reader);
```

vs.

```
cdkReadCommentsFunctionServiceRoleDefaultPolicy067671F9:
  Type: AWS::IAM::Policy
  Properties:
    PolicyDocument:
      Statement:
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
          Effect: Allow
          Resource:
            - Fn::GetAtt:
                - cdkblogcommentsbucket2CABB01E
                - Arn
            - Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - cdkblogcommentsbucket2CABB01E
                      - Arn
                  - /*
      Version: "2012-10-17"
    PolicyName:
cdkReadCommentsFunctionServiceRoleDefaultPolicy067671F9
    Roles:
      - Ref: cdkReadCommentsFunctionServiceRole3D1016B6
```

5 Adding the Lambda to the API Gateway

```
export function createGateway(stack: cdk.Stack,
                              readLambda: NodejsFunction,
                              writeLambda: NodejsFunction): [string, string] {
  const api = new apigateway.RestApi(stack, 'demo-api', {
    description: 'Demo API',
    deployOptions: {
      stageName: 'dev',
    },
    defaultCorsPreflightOptions: {
      allowHeaders: ['Content-Type', 'X-Amz-Date', 'Authorization', 'X-Api-Key'],
      allowMethods: ['OPTIONS', 'GET'],
      allowCredentials: true,
      allowOrigins: ['http://localhost:4200'],
    },
  });
  const Resource = api.root.addResource('').addResource('{url}');
  const readMethod = createMethod(Resource, readLambda, 'GET');
  const writeMethod = createMethod(Resource, writeLambda, 'POST');
  const plan = api.addUsagePlan('MyUsagePlan', {
    name: 'SlowDownHackers',
    throttle: {
      rateLimit: 10,
      burstLimit: 5,
    },
  });
  const apiKey = api.addApiKey('MyApiKey', {
    value: '2734283472839472893 4729',
  });
  plan.addApiKey(apiKey);
  plan.addApiStage({
    stage: api.deploymentStage,
    throttle: [
      {
        method: readMethod,
        throttle: {
          rateLimit: 10,
          burstLimit: 2,
        },
      },
      {
        method: writeMethod,
        throttle: {
          rateLimit: 10,
          burstLimit: 2,
        },
      },
    ],
  });
  new ssm.StringParameter(stack, 'apiGatewayUrlParam', {
    allowedPattern: '.*',
    description: 'apiGatewayUrl',
    parameterName: 'apiGatewayUrlParam',
    stringValue: api.url,
  });

  return ['2734283472839472893 4729', api.restApiId];
}
function createMethod(Resource: apigateway.Resource,
          readLambda: NodejsFunction, httpMethod: 'GET' | 'POST') {
  const readMethod = Resource.addMethod(
    httpMethod,
    new apigateway.LambdaIntegration(readLambda, {
      proxy: true,
    }),
    { apiKeyRequired: true }
  );
  return readMethod;
}
```

6 Adding S3 to CloudFront

```
export function createCloudfrontDistribution(stack: cdk.Stack, bucket: s3.Bucket) {
  const certificate = useExistingCertificateByArn(stack);
  const cf = new cloudfront.Distribution(stack, 'cdk-blog-cdn', {
    defaultBehavior: {
      origin: new origins.S3Origin(bucket),
      cachePolicy: CachePolicy.CACHING_OPTIMIZED,
    },
    domainNames: ['example.com'],
    certificate,
    comment: 'cdk.example.com',
  });
  return cf;
}
```

7 Adding the API Gateway to Cloudfront

```
const cf = new cloudfront.Distribution(stack, 'cdk-blog-cdn', {
  defaultBehavior: {
    origin: new origins.S3Origin(bucket),
    cachePolicy: CachePolicy.CACHING_OPTIMIZED,
  },
  additionalBehaviors: {
    'prod/*': {
      origin: createApiGatewayOrigin(stack, gatewayId, apiKey),
      allowedMethods: AllowedMethods.ALLOW_ALL,
      viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
      cachePolicy: CachePolicy.CACHING_OPTIMIZED,
    },
  },
});

function createApiGatewayOrigin(stack: cdk.Stack, gatewayId: string, apiKey: string):
cloudfront.IOrigin {
  return new origins.HttpOrigin(`${gatewayId}.execute-api.${stack.region}.amazonaws.com`, {
    customHeaders: {
      herkunft: 'CDK Cloudfront Disto',
      'x-api-key': apiKey,
    },
  });
}
```

7 Adding the API Gateway to Cloudfront

```
const cf = new cloudfront.Distribution(stack, 'cdk-blog-cdn', {
  defaultBehavior: {
    origin: new origins.S3Origin(bucket),
    cachePolicy: CachePolicy.CACHING_OPTIMIZED,
  },
  additionalBehaviors: {
    'prod/*': {
      origin: createApiGatewayOrigin(stack, gatewayId, apiKey),
      allowedMethods: AllowedMethods.ALLOW_ALL,
      viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
      cachePolicy: CachePolicy.CACHING_OPTIMIZED,
    },
  },
});

function createApiGatewayOrigin(stack: cdk.Stack, gatewayId: string, apiKey: string):
cloudfront.IOrigin {
  return new origins.HttpOrigin(`${gatewayId}.execute-api.${stack.region}.amazonaws.com`, {
    customHeaders: {
      herkunft: 'CDK Cloudfront Disto',
      'x-api-key': apiKey,
    },
  });
}
```

Surprise!

Slightly confusion documentation

7 Adding the API Gateway to Cloudfront

```
const cf = new cloudfront.Distribution(stack, 'cdk-blog-cdn', {
  defaultBehavior: {
    origin: new origins.S3Origin(bucket),
    cachePolicy: CachePolicy.CACHING_OPTIMIZED,
  },
  additionalBehaviors: {
    'prod/*': {
      origin: createApiGatewayOrigin(stack, gatewayId, apiKey),
      allowedMethods: AllowedMethods.ALLOW_ALL,
      viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
      cachePolicy: CachePolicy.CACHING_OPTIMIZED,
    },
  },
});

function createApiGatewayOrigin(stack: cdk.Stack, gatewayId: string, apiKey: string):
cloudfront.IOrigin {
  return new origins.HttpOrigin(`${gatewayId}.execute-api.${stack.region}.amazonaws.com`, {
    customHeaders: {
      herkunft: 'CDK Cloudfront Disto',
      'x-api-key': apiKey,
    },
  });
}
```

No high-level construct (yet)

need to use a http-based access

8 Cloudwatch, Alarms and Actions

- Cloudfront metrics are only available in us-east-1
- CDK doesn't support cross-region-stacks (yet)
- My fallback solution: AWS CLI glues two stacks

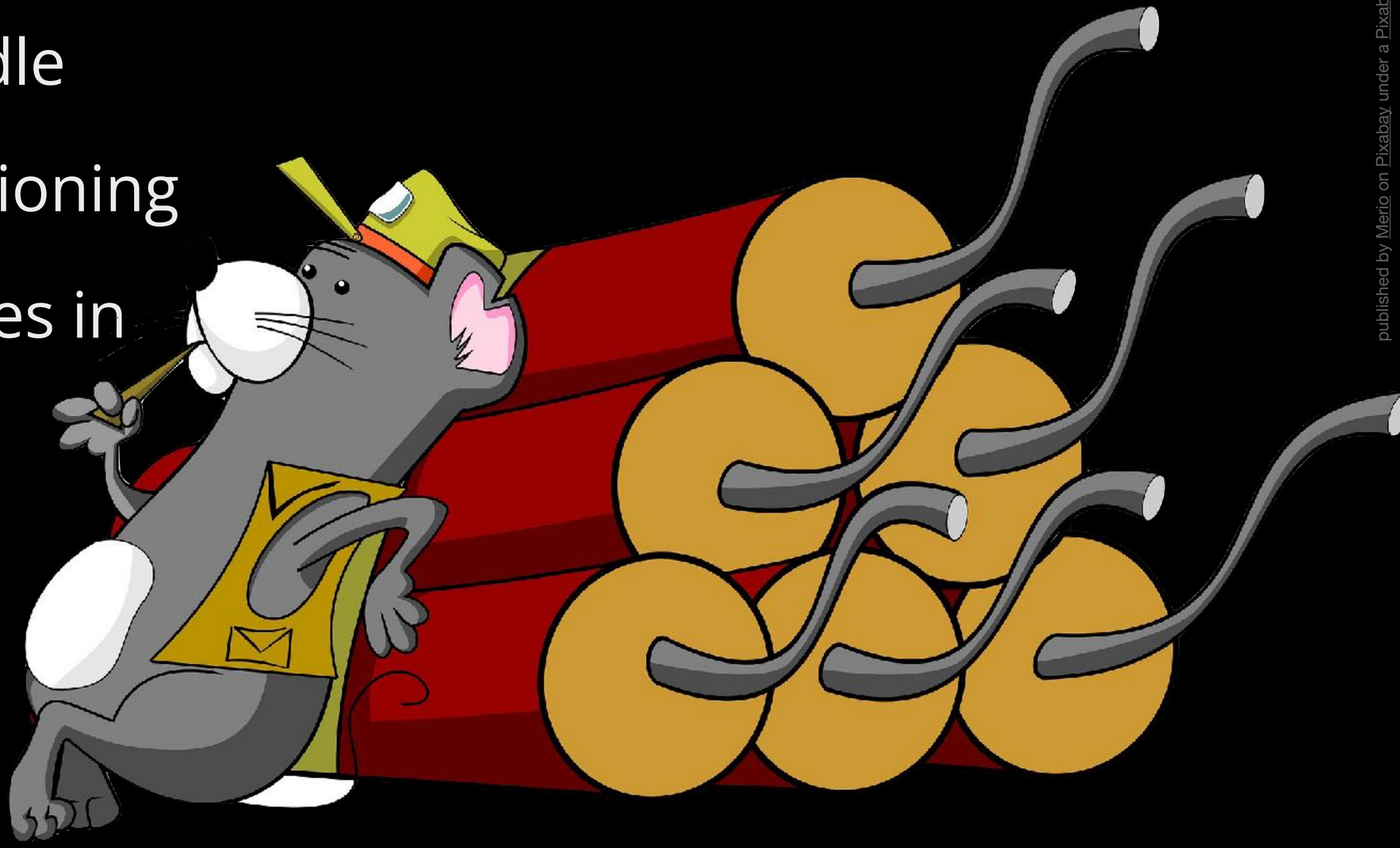
```
run=1 cdk deploy BlogStack
lambdaArnParam=$(aws ssm get-parameter --name lambdaArnParam \
                 --query Parameter.Value --output text)
run=2 lambdaArnParam=$lambdaArnParam mail=$mail cdk deploy CdkAlarmStack
alarmTopic=$(aws --region us-east-1 ssm get-parameter \
             --name alarmTopicArn --query 'Parameter.Value' --output text)
aws --region us-east-1 sns subscribe --topic-arn $alarmTopic \
  --protocol lambda \
  --notification-endpoint $lambdaArnParam
```

9 State of the Art

- Generally speaking: it's awesome!
- Slow deployment
  - use LocalStack instead
  - try `--hotswap`
- Lacking support for cross-region / cross-account stacks
- Documentation / tutorials / blogs don't cover everything
  - … but much better than last year!
- Not every AWS feature is supported yet (still looking for a list!)

10 The Ugly Parts

- Danger ahead: overengeneering!
  - remember Gradle
- No semantic versioning
  - breaking changes in minor versions



Wrapping it Up

- CDK makes a lot of sense for developers
- Ops people won't agree
- Using a proper programming language makes so many things easier
- Code completion, variables, functions, splitting stacks across multiple files, refactoring, sensible defaults, ...
- Self-documenting code helps onboarding and maintenance

Further Reading

all links point to articles, not to the root page:

- https://bobbyhadz.com
- https://blog.dennisokeeffe.com
  (CDN with Route 53 and A-Record)
- https://www.endoflineblog.com by Adam Ruka
- https://blog.codecentric.de
- https://www.syrocon.de

Any Questions?

Don't hesitate to reach out to me!

stephan.rauh@opitz-consulting.com (till 03/25/2022)

stephan.rauh@syrocon.de (from 04/01/2022)

articles@beyondjava.de

@beyondjava

www.beyondjava.net