Reaktor

AWS CDK introduction

AWS Cloud Development Kit

Markus Lindqvist
Senior Solutions Architect
08/2019

Reaktor
Yliopistonkatu 4
00100, Helsinki Finland

tel: +358 9 4152 0200
www.reaktor.com
info@reaktor.com

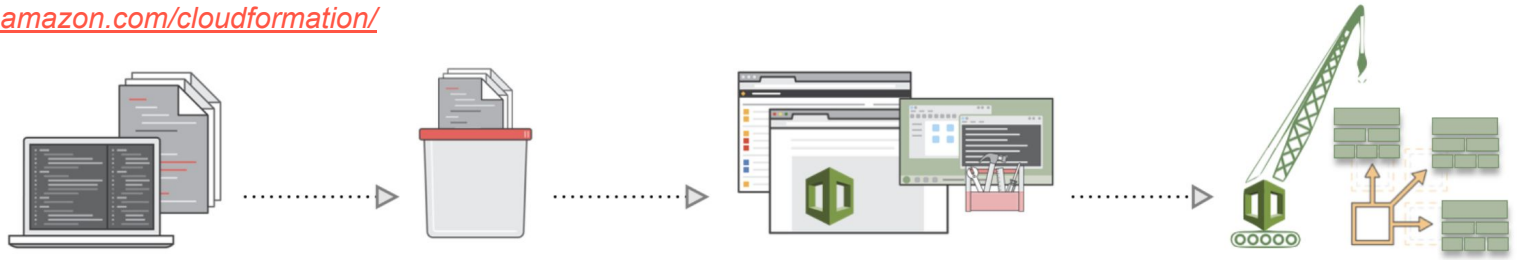
Contents

1. What is AWS CDK
2. Project experiences
3. AWS CDK alternatives
4. Hands-on working with AWS CDK
5. Discussion, Q&A

What is CloudFormation

"AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment"

https://aws.amazon.com/cloudformation/



Code your infrastructure from scratch with the CloudFormation template language, in either YAML or JSON format, or start from many available sample templates

"Code"

Check out your template code locally, or upload it into an S3 bucket

Upload

Use AWS CloudFormation via the browser console, command line tools or APIs to create a stack based on your template code

Create Stack

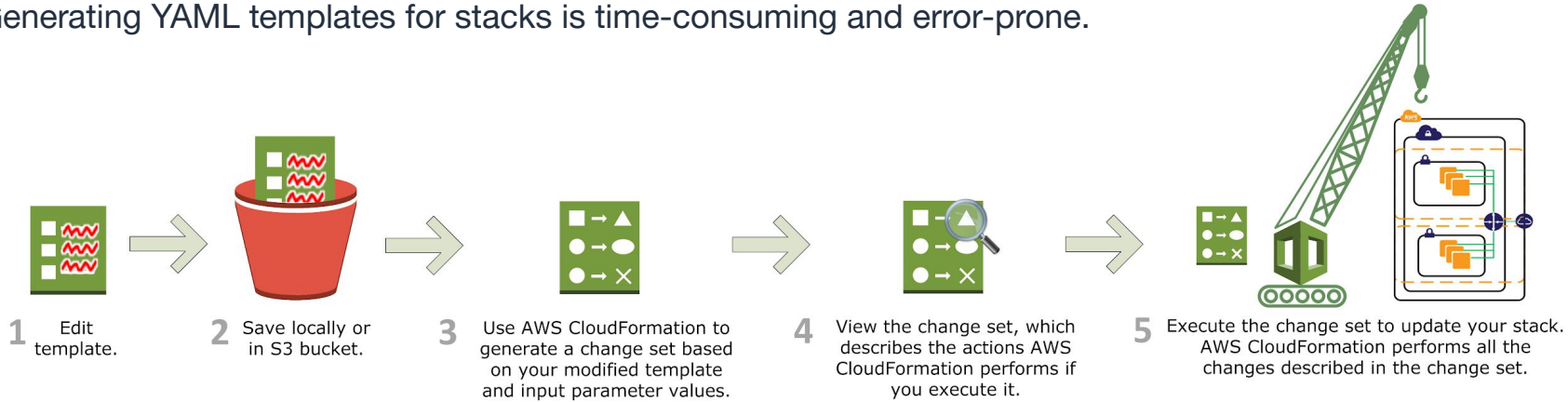
AWS CloudFormation provisions and configures the stacks and resources you specified on your template

CloudFormation workflow

With CloudFormation you work with **Stacks** that contain resources.

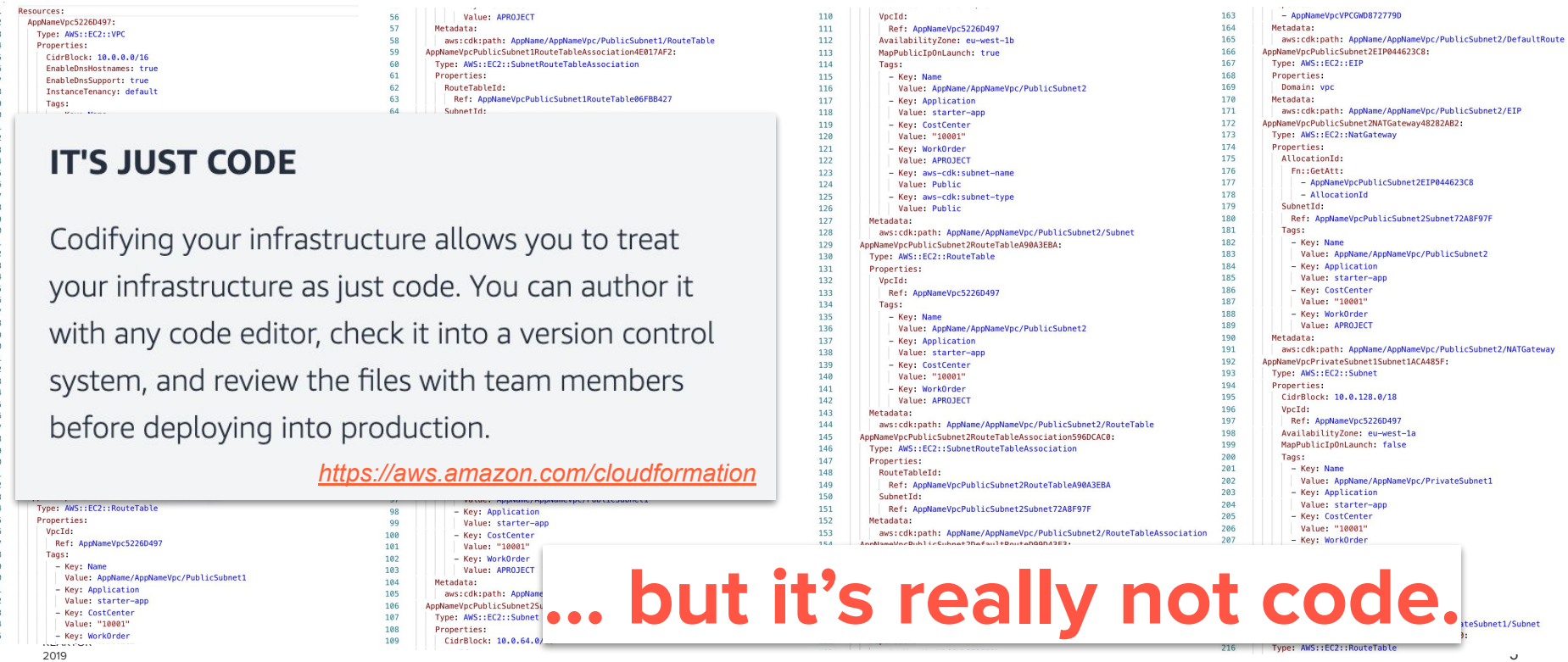
Updating stacks happens via the CLI, Web Console or CodePipeline.

Generating YAML templates for stacks is time-consuming and error-prone.



Source: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-howdoesitwork.html

The problem with CloudFormation

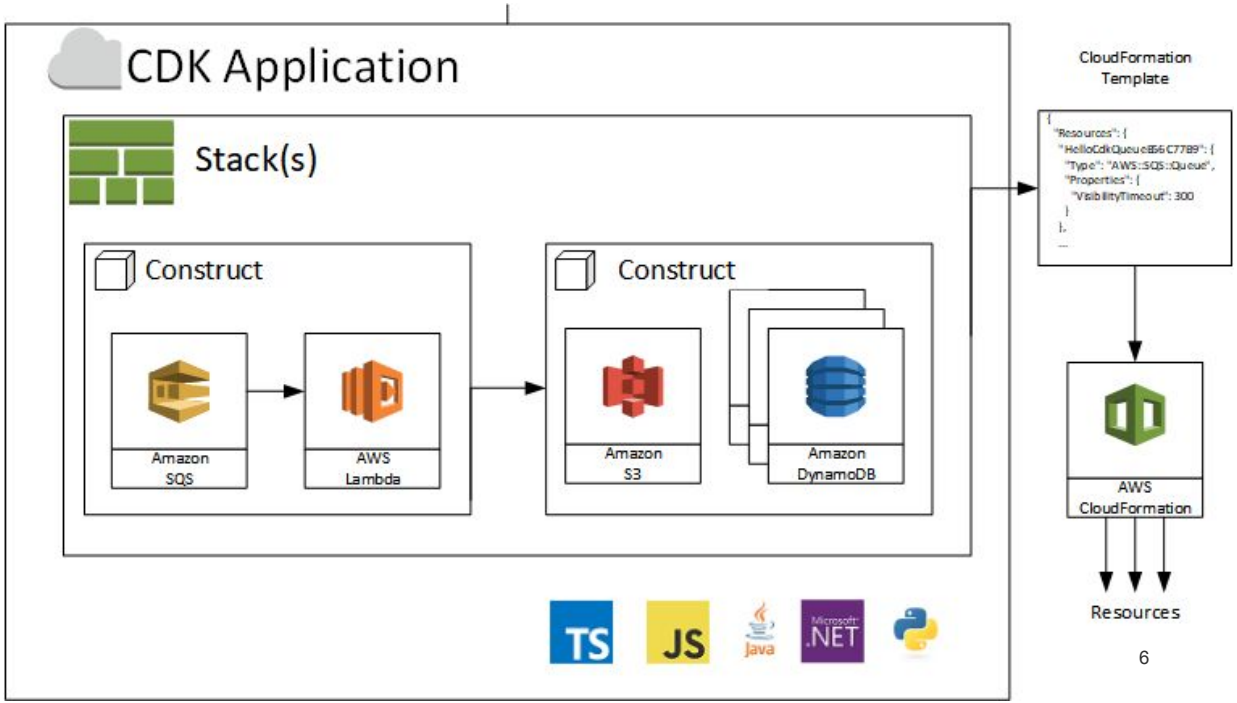


What is AWS CDK?

With AWS CDK you can define CloudFormation resources with multiple programming languages (for example TypeScript).

Concepts of CDK:

- Generate CloudFormation templates
- Deployment tools included
- Aim at creating reusable constructs



Using higher-level languages for producing infra

```
#!/usr/bin/env node
import * as cdk from '@aws-cdk/core';
import * as s3 from '@aws-cdk/aws-s3';

const myApp = new cdk.App();
const stack = new cdk.Stack(myApp, 'BucketStack');
new s3.Bucket(stack, 'ExampleBucket', {
 removalPolicy: cdk.RemovalPolicy.DESTROY,
});
myApp.synth();
```

tack.yaml

```
Resources:
  ExampleBucketDC717CF4:
    Type: AWS::S3::Bucket
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
    Metadata:
      aws:cdk:path: BucketStack/ExampleBucket/Resource
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Modules: aws-cdk=1.0.0,jsii-runtime=node.js/v12.0.0
```

aws-cdk-example.ts → **CDK synth** → Synthesized CloudFormation stack

Use higher-level languages for producing infrastructure templates

```
#!/usr/bin/env node
import * as cdk from '@aws-cd
import * as s3 from '@aws-cdk/a

const myApp = new cdk.App();
const stack = new cdk.Stack(myApp, 'BucketStack');
new s3.Bucket(stack, 'ExampleBucket', {
 removalPolicy: cdk.RemovalPolicy.DESTROY,
});
myApp.synth();
```

All CDK apps have a single ‘app’

An app can have multiple stacks

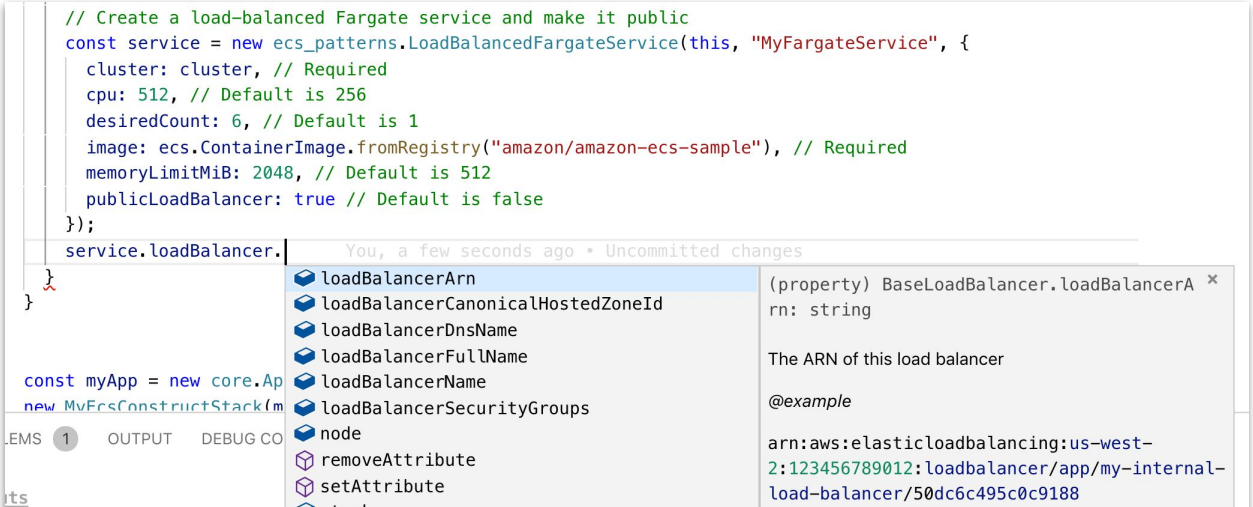
Different resources are instantiated in stacks

```
yaml
                                    y: Delete
DeletionPolicy: Delete
Metadata:
  aws:cdk:path: BucketStack/ExampleBucket/Resource

                                    etadata

                                    s-cdk=1.0.0,jsii-runtime=node.js/v12.0.0
```

aws-cdk-example.ts → **CDK synth** → Synthesized CloudFormation stack

Main advantages of AWS CDK

- AWS CDK has IDE-support (due to using real programming languages) and good tooling
  - This means code completion and inline documentation!
- AWS CDK is as human readable as code
- Easier to reuse higher-level constructs
  - Use existing infra for sharing components (NPM)
- Deployment methods
- Open source
- AWS official tools



Useful links

- https://docs.aws.amazon.com/cdk/latest/guide/getting_started.html
- https://docs.aws.amazon.com/cdk/api/latest/docs/aws-construct-library.html
- https://github.com/aws-samples/aws-cdk-examples
- https://github.com/aws/aws-cdk

Project experiences

Experiences from real-life projects

- 👍 It’s extremely easy to start using AWS CDK
  - Using existing CloudFormation templates and Terraform tools works well side-by-side with AWS CDK
  - AWS CDK can easily import CloudFormation/Terraform created resources
  - Creating and deploying standalone stacks is straight forward
- 👍 AWS CDK is constructed using commonly known tools (Node etc.)
- 👍 Defaults to well-defined resource-based IAM roles
- 👍 High-quality toolset: Well written documentation, good release notes
- 👍 Open source development model
  - Fast feedback for bug reports on Github
  - User submitted PR’s are handled quickly and professionally
  - https://github.com/aws/aws-cdk/pulls

Experiences from real-life projects

- 👎 Sometimes stacks fail only when deployed
  - The emitted templates seem to be valid all the time
  - Validation for certain aspects could be done already at CDK level (see next slides)
- 👎 Generated templates more verbose than hand-written ones
- 👎 CloudFormation does not support all AWS resources
  - Not really a CDK limitation
  - For example, we provision CloudHSM instances using Terraform
- 👎 Additional abstraction
  - Can produce errors from multiple layers
- 👎 Continuous Delivery model still in progress
  - https://github.com/aws/aws-cdk/pull/3437

Spot two errors

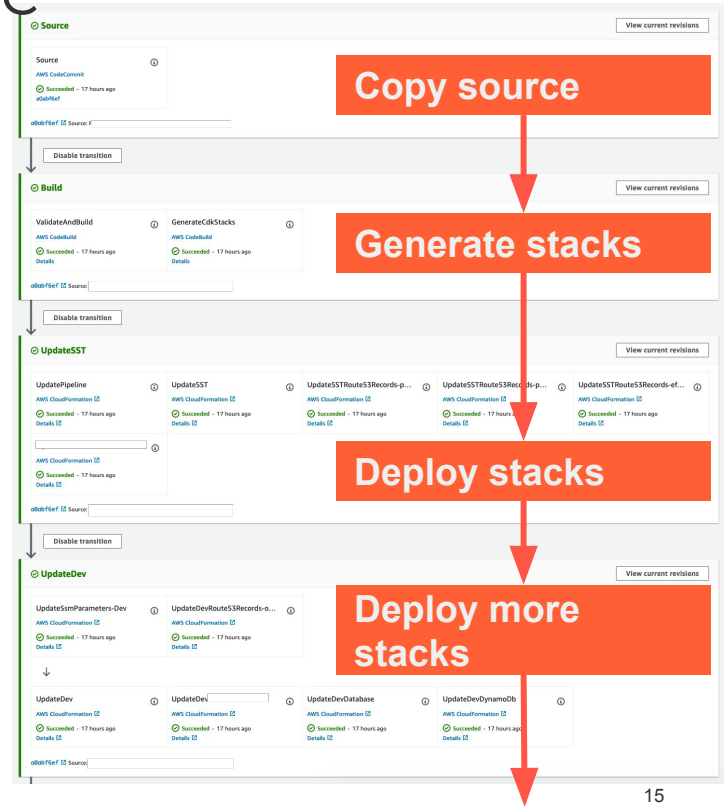
- The errors unfortunately appear only at deployment time
  - User would prefer predeployment validation if possible

```
const policy = new iam.PolicyStatement({
    actions: ['s3:GetObject'],
    resources: ['*'],
    principals: [new iam.AnyPrincipal()],
    effect: iam.Effect.ALLOW,
    conditions: [
        'IpAddress', {
            'aws:sourceIp': [source]
        }]
});
```

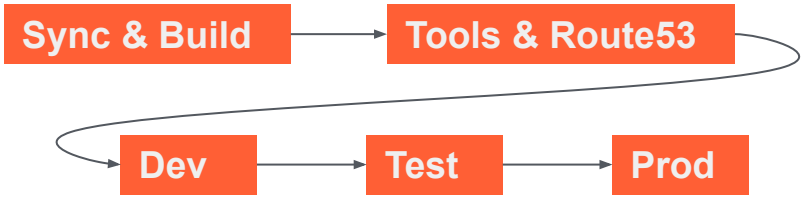
```
const policy = new iam.PolicyStatement({
    actions: ['s3:GetObject'],
    resources: [bucket.arnForObjects('*')],
    principals: [new iam.AnyPrincipal()],
    effect: iam.Effect.ALLOW,
    conditions: {
        'IpAddress': {
            'aws:sourceIp': [source]
        }}
});
```

Experiences: AWS CodePipeline

- Use CodePipeline to synthetize the CDK stacks
- New CDK stacks are deployed in parallel to existing CloudFormation stacks
  - We are using similar cross-account deployment methods with CF and CDK stacks



Deployment model



CodePipeline buildspec.yaml

- Using standard CodePipeline tools and templates
- Nodejs base image
- Standalone templates can easily be synthesized during a CodePipeline build
- Generated templates are applied later in the build process with predefined cross-account roles

```yaml
phases:
 install:
   runtime-versions:
     nodejs: 10
   commands:
     - cd cdk
     - npm i
     - npm i -g aws-cdk@1.3.0
 pre_build:
   commands:
     - npm run build
 build:
   commands:
     # Synthesize CDK templates
     - cdk synth app-route53 > app-route53-template.yaml
     - cdk synth app-dev-stack1 > app-dev-stack1-template.yaml
     - cdk synth app-test-stack1 > app-test-stack1-template.yaml
     - cdk synth app-prod-stack1 > app-prod-stack1-template.yaml
```

Deploying Lambdas with CDK

- Previously we used the Serverless Framework and AWS SAM to deploy Lambda functions to perform simple tasks
- These were refactored to AWS CDK as well ➜ less technologies, less complexity
- Slightly more verbose than pure serverless configuration

```
const updateLambda = new lambda.Function(stack, FunctionName, {
    functionName: FunctionName,
    description: FunctionDescription,
    runtime: lambda.Runtime.NODEJS_8_10,
    handler: 'index.handler',
    role: lambdaRole,
    code: lambda.Code.inline(
        fs.readFileSync('./update-cluster-lambda/index.js').toString()),
    environment: {
        TARGET_CLUSTER: `${env}-cluster`,
        TARGET_IMAGE_TAG: tag,
    }
});
```

Using unsupported CloudFormation constructs

- Example: At the moment there is no redirect support for ALB's in CDK
- By using raw CloudFormation constructs, we can easily achieve the needed functionality
- https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-listener.html
- https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-elasticloadbalancingv2-listener-redirectconfig.html

```
const createHttpsRedirect = (id: string, scope: cdk.Construct, loadBalancer:
elbv2.ApplicationLoadBalancer) => {
 const actionProperty: elbv2.CfnListener.ActionProperty = {
   type: 'redirect',
   redirectConfig: {
     statusCode: 'HTTP_302',
     protocol: 'HTTPS',
     port: '443',
   },
 };
 const redirectProps: elbv2.CfnListenerProps = {
   defaultActions: [actionProperty],
   loadBalancerArn: loadBalancer.loadBalancerArn,
   port: 80,
   protocol: 'HTTP',
 };
 return new elbv2.CfnListener(scope, `${id}HttpRedirect`, redirectProps);
};
```

AWS CDK alternatives

Alternatives to AWS CDK

Solution	Language	Comments
CloudFormation from AWS	YAML	Long, repetitive definition files that most likely need scripting in real-world use cases.
HashiCorp Terraform	HCL	Multi-cloud solution with custom “programming language” including deployment tools. Paid option available.
Pulumi from Pulumi	TypeScript	Multi-cloud devops solution with deployment automation. Still in early stages of development. Paid option available.
Troposphere	Python	Widely-used Python library for generating CloudFormation templates
Serverless	YAML	Multi-cloud deployment & development tool for serverless software
Boto3 from AWS	Python	AWS’s own Python SDK. Can be used to eg. automate stack creation
Cloudform	TypeScript	Small hobby-looking project for creating AWS resources with TypeScript.

Terraform

- Partial Terraform template for provisioning an EC2 instance
- Source: https://github.com/terraform-aws-modules/terraform-aws-ec2-instance/tree/master/examples/basic

```
module "ec2" {
 source = "../../"
 instance_count = 2
 name          = "example-normal"
 ami           = data.aws_ami.amazon_linux.id
 instance_type = "c5.large"
 subnet_id     = tolist(data.aws_subnet_ids.all.ids)[0]
 //  private_ips                 = ["172.31.32.5", "172.31.46.20"]
 vpc_security_group_ids      =
[module.security_group.this_security_group_id]
 associate_public_ip_address = true
 placement_group             = aws_placement_group.web.id
 root_block_device = [
   {
     volume_type = "gp2"
     volume_size = 10
   },
 ]
```

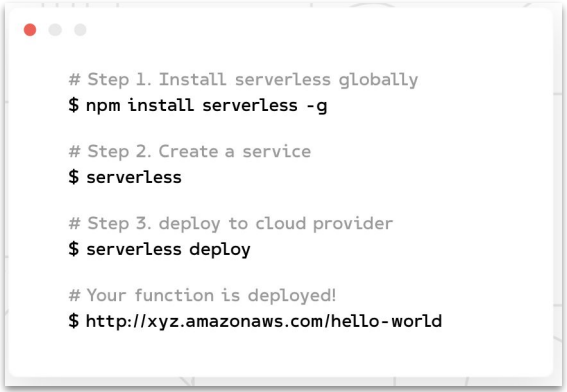
Pulumi

- Partial Pulumi template for provisioning an EC2 instance
- Source: https://github.com/pulumi/examples/tree/master/aws-js-webserver

```
// create a new security group for port 80
let group = new aws.ec2.SecurityGroup("web-secgrp", {
   ingress: [
       { protocol: "tcp", fromPort: 22, toPort: 22, cidrBlocks:
["0.0.0.0/0"] },
       { protocol: "tcp", fromPort: 80, toPort: 80, cidrBlocks:
["0.0.0.0/0"] },
   ],
});

let server = new aws.ec2.Instance("web-server-www", {
   tags: { "Name": "web-server-www" },
   instanceType: size,
   securityGroups: [ group.name ], // reference the group object
above
   ami: ami,
   userData: userData              // start a simple web server
});
```

Serverless Framework

- Example of a Serverless deployment file for a NodeJs application



https://serverless.com/framework/

```yaml

service: aws-node-auth0-cognito-custom-authorizers-api

provider:
  name: aws
  runtime: nodejs8.10

functions:
  publicEndpoint:
    handler: handler.publicEndpoint
    events:
      - http:
          path: api/public
          method: get
          integration: lambda
          cors: true
  auth:
    handler: auth.authorize
  privateEndpoint:
    handler: handler.privateEndpoint
    events:
      - http:
          path: api/private
          method: get
          authorizer: auth
          cors:
            origins:
              - '*'
            headers:
              - Content-Type
              - X-Amz-Date
```


Source: https://github.com/serverless/examples/blob/master/aws-node-auth0-cognito-custom-authorizers-api/serverless.yml

troposphere

- A Python library for generating CloudFormation templates
- Does not have deployment tools
- Supports many AWS and OpenStack resource types
- See https://github.com/cloudtools/troposphere

A simple example to create an instance (YAML) would look like this:

```
>>> from troposphere import Ref, Template
>>> import troposphere.ec2 as ec2
>>> t = Template()
>>> instance = ec2.Instance("myinstance")
>>> instance.ImageId = "ami-951945d0"
>>> instance.InstanceType = "t1.micro"
>>> t.add_resource(instance)
<troposphere.ec2.Instance object at 0x101bf3390>
>>> print(t.to_yaml())

Resources:
    myinstance:
        Properties:
            ImageId: ami-951945d0
            InstanceType: t1.micro
        Type: AWS::EC2::Instance
```

AWS CDK hands-on

Part 1 - the simplest possible stack

REAKTOR
2019

Install AWS CDK and get started

```
npm install -g aws-cdk
mkdir aws-cdk-example
cd aws-cdk-example
cdk init --language typescript
```

This will give you the following output

```
Applying project template app for typescript
Initializing a new git repository...
Executing npm install...

# Useful commands

 * `npm run build`   compile typescript to js
 * `npm run watch`   watch for changes and compile
 * `cdk deploy`      deploy this stack to your default AWS account/region
 * `cdk diff`        compare deployed stack with current state
 * `cdk synth`       emits the synthesized CloudFormation template
```

Try out the CDK commands

The outputs are the **stack name** and YAML file for an **empty stack** that contains the CDK metadata.

```
mml@l1152~/aws-cdk-example $ cdk ls
AwsCdkExampleStack
mml@l1152~/aws-cdk-example $ cdk synth
Resources:
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Modules: aws-cdk=1.0.0,jsii-runtime=node.js/v12.0.0
```

Create your first app

Edit the `bin/aws-cdk-example.ts` with the following content **https://bit.ly/33sgFlU**

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from '@aws-cdk/core';
import * as s3 from '@aws-cdk/aws-s3';

const myApp = new cdk.App();
const stack = new cdk.Stack(myApp, 'BucketStack');
new s3.Bucket(stack, 'ExampleBucket', {
 removalPolicy: cdk.RemovalPolicy.DESTROY,
});
myApp.synth();
```

Add S3 dependency and check that everything builds

```
npm install --save-dev @aws-cdk/aws-s3
npm run build
```

Deploying the stack

Run the following command to deploy the stack

```
cdk deploy --profile demo BucketStack
```

See documentation for credential setup
https://docs.aws.amazon.com/cdk/latest/guide/getting_started.html#getting_started_credentials

```
BucketStack: deploying...
BucketStack: creating CloudFormation changeset...
 0/3 | 9:53:18 PM | REVIEW_IN_PROGRESS   | AWS::CloudFormation::Stack | BucketStack User Initiated
 0/3 | 9:53:23 PM | CREATE_IN_PROGRESS   | AWS::CloudFormation::Stack | BucketStack User Initiated
 0/3 | 9:53:27 PM | CREATE_IN_PROGRESS   | AWS::CDK::Metadata | CDKMetadata
 0/3 | 9:53:27 PM | CREATE_IN_PROGRESS   | AWS::S3::Bucket    | ExampleBucket (ExampleBucketDC717CF4)
 0/3 | 9:53:28 PM | CREATE_IN_PROGRESS   | AWS::S3::Bucket    | ExampleBucket (ExampleBucketDC717CF4) Resource creation Initiated
 0/3 | 9:53:29 PM | CREATE_IN_PROGRESS   | AWS::CDK::Metadata | CDKMetadata Resource creation Initiated
 1/3 | 9:53:29 PM | CREATE_COMPLETE      | AWS::CDK::Metadata | CDKMetadata
 2/3 | 9:53:49 PM | CREATE_COMPLETE      | AWS::S3::Bucket    | ExampleBucket (ExampleBucketDC717CF4)
 3/3 | 9:53:51 PM | CREATE_COMPLETE      | AWS::CloudFormation::Stack | BucketStack

✅ BucketStack

Stack ARN:
arn:aws:cloudformation:eu-west-1:872821666058:stack/BucketStack/f980c820-a7fa-11e9-80b3-02f48c625dae
```

Updating the stack

First, modify a resource created by the stack and the run a command to diff against deployed version.

```
cdk diff --profile demo
```

```
Stack BucketStack
Resources
[~] AWS::S3::Bucket ExampleBucket ExampleBucketDC717CF4
 └─ [+] BucketEncryption
     └─ {"ServerSideEncryptionConfiguration":[{"ServerSideEncryptionByDefault":{"SSEAlgorithm":"AES256"}}]}
```

```
6  const myApp = new cdk.App();
7  const stack = new cdk.Stack(myApp, 'BucketStack');
8  new s3.Bucket(stack, 'ExampleBucket', {
9    removalPolicy: cdk.RemovalPolicy.DESTROY,
10   encryption: s3.BucketEncryption.S3_MANAGED,
11 });      You, a few seconds ago • Uncommitted change
12 myApp.synth();
```

Doing deployment works as expected.

```
BucketStack: deploying...
BucketStack: creating CloudFormation changeset...
 0/2 | 10:01:41 PM | UPDATE_IN_PROGRESS    | AWS::CloudFormation::Stack | BucketStack User Initiated
 0/2 | 10:01:45 PM | UPDATE_IN_PROGRESS    | AWS::S3::Bucket    | ExampleBucket (ExampleBucketDC717CF4)
 1/2 | 10:02:06 PM | UPDATE_COMPLETE       | AWS::S3::Bucket    | ExampleBucket (ExampleBucketDC717CF4)
 1/2 | 10:02:08 PM | UPDATE_COMPLETE_CLEA | AWS::CloudFormation::Stack | BucketStack
 2/2 | 10:02:08 PM | UPDATE_COMPLETE       | AWS::CloudFormation::Stack | BucketStack

✅ BucketStack
```

How it looks in the AWS console

Note the difference to Terraform deployments: All created resources are associated with a stack.

All stacks have modification history in the AWS Console.

Stacks can be checked for differences using AWS tools (“drift detection”).

Logical ID	Status	Type
BucketStack	UPDATE_COMPLETE	AWS::CloudFormation::Stack
BucketStack	UPDATE_COMPLETE_CLEANUP_IN_PROGRESS	AWS::CloudFormation::Stack
ExampleBucketDC717CF4	UPDATE_COMPLETE	AWS::S3::Bucket
ExampleBucketDC717CF4	UPDATE_IN_PROGRESS	AWS::S3::Bucket
BucketStack	UPDATE_IN_PROGRESS	AWS::CloudFormation::Stack
BucketStack	CREATE_COMPLETE	AWS::CloudFormation::Stack
ExampleBucketDC717CF4	CREATE_COMPLETE	AWS::S3::Bucket
CDKMetadata	CREATE_COMPLETE	AWS::CDK::Metadata
CDKMetadata	CREATE_IN_PROGRESS	AWS::CDK::Metadata
ExampleBucketDC717CF4	CREATE_IN_PROGRESS	AWS::S3::Bucket
ExampleBucketDC717CF4	CREATE_IN_PROGRESS	AWS::S3::Bucket
CDKMetadata	CREATE_IN_PROGRESS	AWS::CDK::Metadata
BucketStack	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack
BucketStack	REVIEW_IN_PROGRESS	AWS::CloudFormation::Stack

AWS CDK hands-on

Part 2 - the simplest possible Fargate cluster

REAKTOR
2019

The easiest way of deploying a Fargate service

- In this example we use AWS sample code to deploy the simplest possible Fargate cluster
- Copy the Fargate example from https://bit.ly/2OODE7e
- Prepare the deployment:
  - `npm install`
  - `npm run build`
  - `cdk deploy --profile demo`

```
const vpc = new ec2.Vpc(this, "MyVpc", {
  maxAzs: 3 // Default is all AZs in region
});

const cluster = new ecs.Cluster(this, "MyCluster", {
  vpc: vpc
});

// Create a load-balanced Fargate service and make it public
const service = new ecs_patterns.LoadBalancedFargateService(this,
"MyFargateService", {
  cluster: cluster, // Required
  cpu: 512, // Default is 256
  desiredCount: 6, // Default is 1
  image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"), // Re
  memoryLimitMiB: 2048, // Default is 512
  publicLoadBalancer: true // Default is false
});
```

From https://docs.aws.amazon.com/cdk/latest/guide/home.html

The easiest way of deploying a Fargate service

- If the changeset contains IAM changes, the CDK toolkit asks you to confirm the IAM changes separately
- This is done for security reasons

IAM Statement Changes

	Resource	Effect	Action	Principal	Condition
+	${MyFargateService/TaskDef/ExecutionRole.Arn}	Allow	sts:AssumeRole	Service:ecs-tasks.${AWS::URLSuffix}	
+	${MyFargateService/TaskDef/TaskRole.Arn}	Allow	sts:AssumeRole	Service:ecs-tasks.${AWS::URLSuffix}	
+	${MyFargateService/TaskDef/web/LogGroup.Arn}	Allow	logs:CreateLogStream logs:PutLogEvents	AWS:${MyFargateService/TaskDef/ExecutionRole}	

Security Group Changes

	Group	Dir	Protocol	Peer
+ +	${MyFargateService/LB/SecurityGroup.GroupId} ${MyFargateService/LB/SecurityGroup.GroupId}	In Out	TCP 80 TCP 80	Everyone (IPv4) ${MyFargateService/Service/SecurityGroup.GroupId}
+ +	${MyFargateService/Service/SecurityGroup.GroupId} ${MyFargateService/Service/SecurityGroup.GroupId}	In Out	TCP 80 Everything	${MyFargateService/LB/SecurityGroup.GroupId} Everyone (IPv4)

(NOTE: There may be security-related changes not in this list. See http://bit.ly/cdk-2EhF7Np)

The easiest way of deploying a Fargate service

IAM Statement Changes

	Resource	Effect	Action	Principal	Condition
+	${MyFargateService/TaskDef/ExecutionRole.Arn}	Allow	sts:AssumeRole	Service:ecs-tasks.${AWS::URLSuffix}	
+	${MyFargateService/TaskDef/TaskRole.Arn}	Allow	sts:AssumeRole	Service:ecs-tasks.${AWS::URLSuffix}	
+	${MyFargateService/TaskDef/web/LogGroup.Arn}	Allow	logs:CreateLogStream logs:PutLogEvents	AWS:${MyFargateService/TaskDef/ExecutionRole}	

Security Group Changes

	Group	Dir	Protocol	Peer
+	${MyFargateService/LB/SecurityGroup.GroupId}	In	TCP 80	Everyone (IPv4)
+	${MyFargateService/LB/SecurityGroup.GroupId}	Out		e/Service/SecurityGroup.GroupId
+	${MyFargateService/Service/SecurityGroup.GroupId}			curityGroup.GroupId}
+	${MyFargateService/Service/SecurityGroup.GroupId}	Out		

(NOTE: There may be security-related changes not in this list. See http://bit.ly/cdk-2EhF7Np)

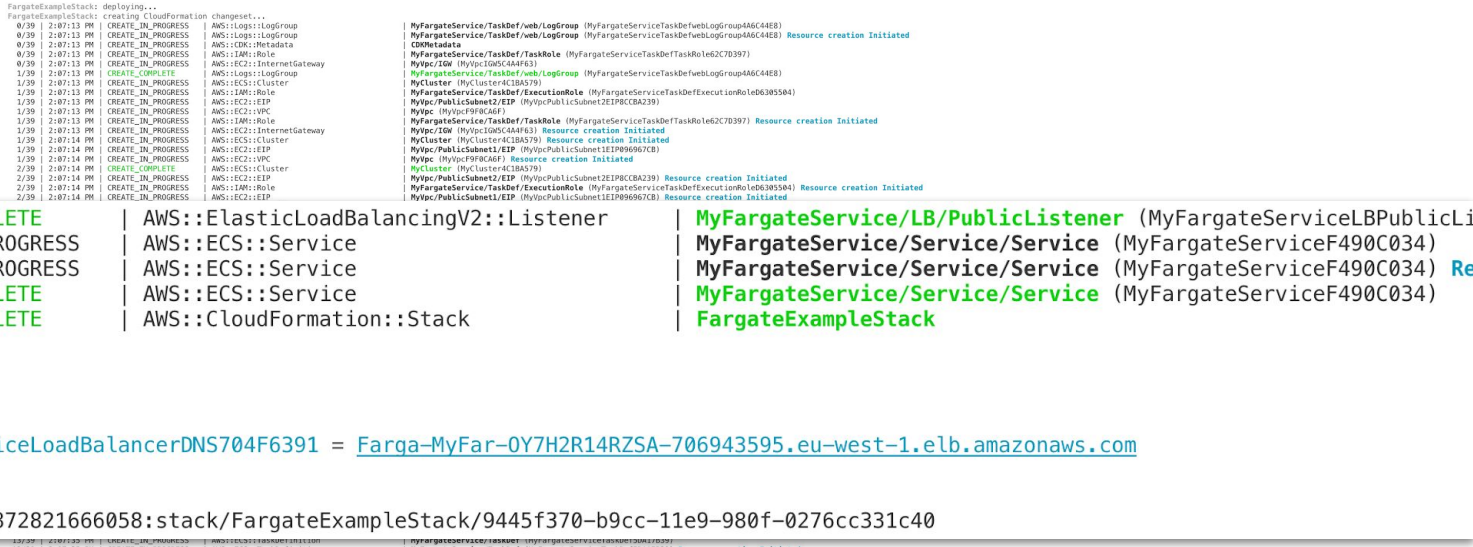
Confirm changes with 'y'

```
Do you wish to deploy these changes (y/n)? y
FargateExampleStack: deploying...
FargateExampleStack: creating CloudFormation changeset...
  0/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::Logs::LogGroup                | /web/LogGroup (MyFargateServiceTaskDefwebLogGroup4A6C44E8)
  0/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::Logs::LogGroup                | /LogGroup (MyFargateServiceTaskDefwebLogGroup4A6C44E8) Resource cre
ation Initiated
  0/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::CDK::Metadata
  0/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role                     | skDef/TaskRole (MyFargateServiceTaskDefTaskRole62C7D397)
  0/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::EC2::InternetGateway          | MyVpc/IGW (MyVpcIGW5C4A4F63)
  1/39 | 2:07:13 PM | CREATE_COMPLETE      | AWS::Logs::LogGroup                | MyFargateService/TaskDef/web/LogGroup (MyFargateServiceTaskDefwebLogGroup4A6C44E8)
  1/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::ECS::Cluster                  | MyCluster (MyCluster4C1BA579)
  1/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role                     | MyFargateService/TaskDef/ExecutionRole (MyFargateServiceTaskDefExecutionRoleD6305504)
  1/39 | 2:07:13 PM | CREATE_IN_PROGRESS   | AWS::EC2::EIP                      | MyVpc/PublicSubnet2/EIP (MyVpcPublicSubnet2EIP8CCBA239)
```

The deployment log will follow

The easiest way of deploying a Fargate service

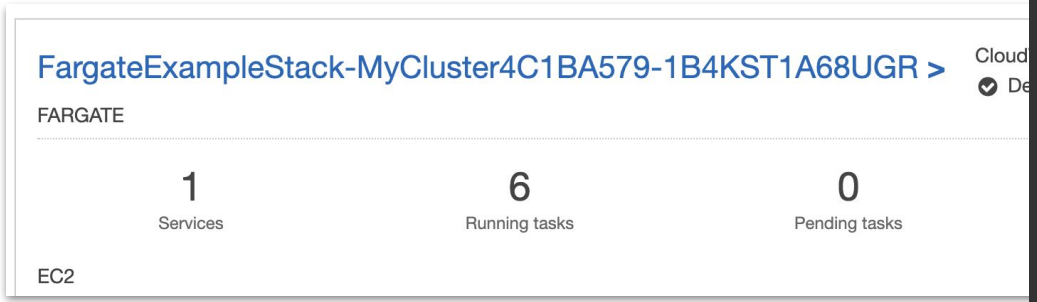
A huge list of changes will be deployed!

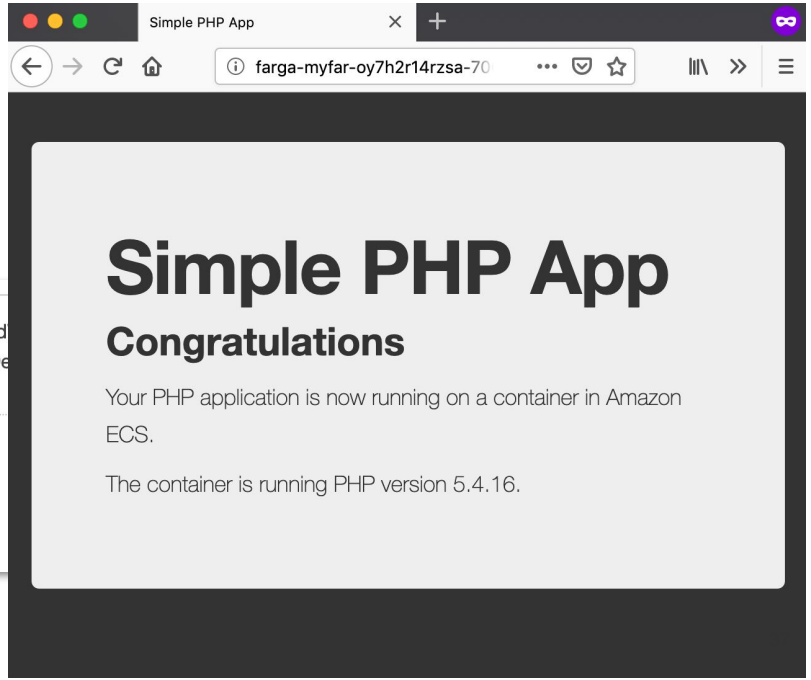


The easiest way of deploying a Fargate service

Check the deployment output. You will see the publicly accessible load balancer URL where the service is running.

The deployed Fargate cluster can be viewed in the AWS console.

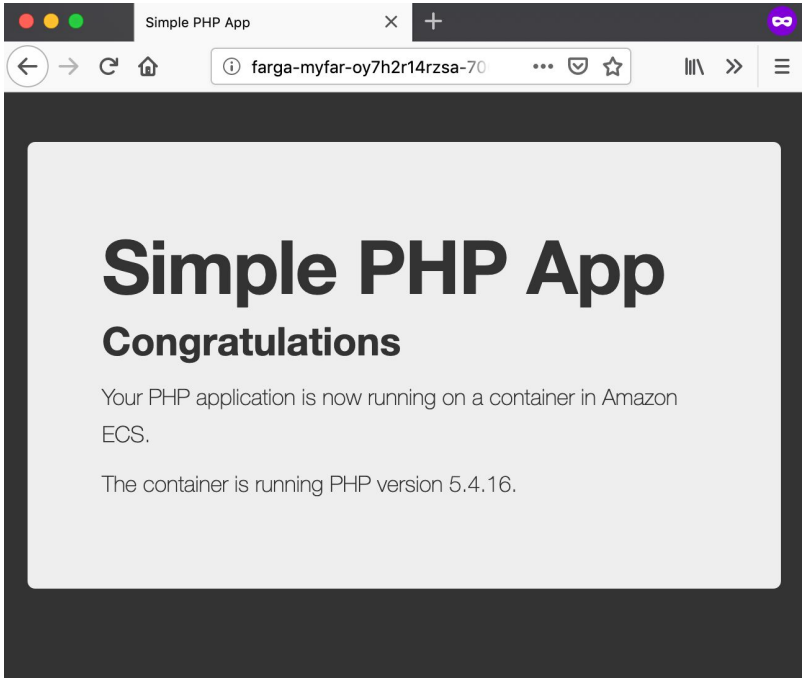




The easiest way of deploying a Fargate service

Avoid unexpected costs by removing the service after it's no longer needed.

```
cdk destroy --profile demo
```



AWS CDK hands-on

Part 3 - Slightly more complex Fargate cluster :-)

REAKTOR
2019

Deploying a version of Fargate infrastructure for hosting API microservices*

A more complex Fargate hosting example using multiple Docker containers and environments can be found at https://github.com/markusl/cdk-fargate-docker-starter

```
git clone git@github.com:markusl/cdk-fargate-docker-starter.git

# Follow README instructions

cdk deploy -c
certificateIdentifier=e3da8de9-ec36-4c75-addd-cc62701eac3a -c
domainName=olmi.be -c subdomainName=site-dev -c environment=dev
```

*Replace the example Docker images with our microservices

AWS CDK: Deploying a stack with assets

You will see an error message if you did not bootstrap the deployment environment first.

After bootstrapping (just once), continue deployment normally.

```
❌ AppName failed: Error: This stack uses assets, so the toolkit stack must be deployed to the environment (Run "cdk bootstrap aws://
872821666058/eu-west-1")
    at Object.prepareAssets (/Users/mml/.nvm/versions/node/v12.0.0/lib/node_modules/aws-cdk/lib/assets.ts:24:11)
    at Object.deployStack (/Users/mml/.nvm/versions/node/v12.0.0/lib/node_modules/aws-cdk/lib/api/deploy-stack.ts:49:24)
```

```
⏳ Bootstrapping environment aws://872821666058/eu-west-1...
CDKToolkit: creating CloudFormation changeset...
 0/2 | 6:23:14 PM | REVIEW_IN_PROGRESS   | AWS::CloudFormation::Stack | CDKToolkit User Initiated
 0/2 | 6:23:16 PM | CREATE_IN_PROGRESS   | AWS::CloudFormation::Stack | CDKToolkit User Initiated
 0/2 | 6:23:19 PM | CREATE_IN_PROGRESS   | AWS::S3::Bucket | StagingBucket
 0/2 | 6:23:20 PM | CREATE_IN_PROGRESS   | AWS::S3::Bucket | StagingBucket Resource creation Initiated
 1/2 | 6:23:41 PM | CREATE_COMPLETE      | AWS::S3::Bucket | StagingBucket
 2/2 | 6:23:42 PM | CREATE_COMPLETE      | AWS::CloudFormation::Stack | CDKToolkit
✅ Environment aws://872821666058/eu-west-1 bootstrapped.
```

Deploying a version of Fargate infrastructure for hosting API microservices

The app contains following features:

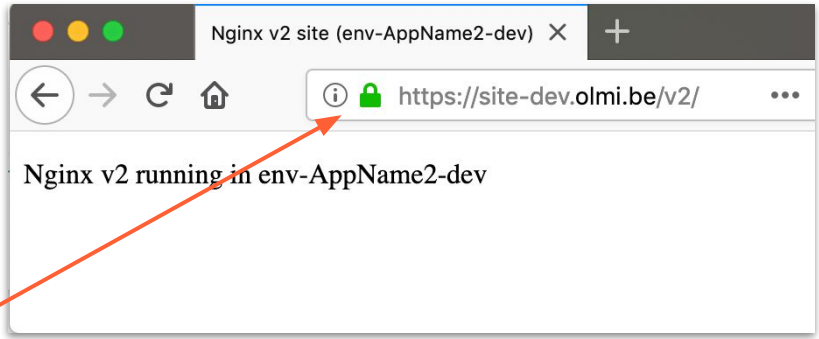
- Logging, tagging
- Host / path matching
- TLS certificate
- Multiple containers
- Configurable microservices
  - See `site-config-dev.ts / site-config-prod.ts`

```
export const dockerProperties: ContainerProperties[] = [
  {
    image: ecs.ContainerImage.fromAsset(containerDirectory),
    containerPort: 80,
    id: 'Service1Name',
    hostHeader: 'site-dev.olmi.be',
    pathPattern: '/service1*',
    environment: { APP_ENVIRONMENT: `env-Service1Name-dev` },
  },
  {
    image: ecs.ContainerImage.fromAsset(containerDirectory),
    containerPort: 80,
    id: 'Service2Name',
    hostHeader: 'site-dev.olmi.be',
    pathPattern: '/service2*',
    environment: { APP_ENVIRONMENT: `env-Service2Name-dev` },
```

Verify the deployment

You can visit the deployed endpoints after the deployment is finished (will take some time as it deploys).

Again, remember to remove the stack after it's not needed.




```
 64/69 | 5:17:40 PM | CREATE_COMPLETE      | AWS::EC2::Route                          | AppName-devVpc/PrivateSubnet2/DefaultRoute (AppNamedevVpcPrivat
 65/69 | 5:17:48 PM | CREATE_COMPLETE      | AWS::Route53::RecordSet                  | AppName-devSite (AppNamedevSite5A7EE027)
65/69 Currently in progress: EcsSampleFargateServiceB962A6EA, AppName2FargateService5AD4B4D4, AppName1FargateService12EF8A56
 66/69 | 5:18:26 PM | CREATE_COMPLETE      | AWS::ECS::Service                        | EcsSampleFargateService/Service (EcsSampleFargateServiceB962A6E
 67/69 | 5:18:27 PM | CREATE_COMPLETE      | AWS::ECS::Service                        | AppName2FargateService/Service (AppName2FargateService5AD4B4D4)
 68/69 | 5:18:27 PM | CREATE_COMPLETE      | AWS::ECS::Service                        | AppName1FargateService/Service (AppName1FargateService12EF8A56)

 ✅  AppName-dev

Outputs:
AppName-dev.SiteDNS = site-dev.olmi.be
AppName-dev.AppNamedevDNS = AppNa-AppNa-1J4YK3XDEZAVC-378028897.eu-west-1.elb.amazonaws.com
```

Discussion

Q&A

Thank you.

markus.lindqvist@reaktor.com