

# Deploying Containerized Web and Serverless Applications using the Cloud Development Kit (CDK)

November 25, 2019

<https://github.com/jvargh/aws-cdk-workshop>

Joji Varghese



# Quick Facts

## Introduction to Infrastructure as Code (IaC)

- What is IaC? Why use it?
  - *Tool to create Cloud resources on your AWS account with minimal manual effort*
- Where does AWS CDK fit into the IaC space?
  - *CDK allows you to use IaC in a programming language of your choice*
  - *CDK allows creation of higher level constructs that create lower level resources on your account*
- What does AWS CDK offer that is unique?
  - *CDK provides built-in Helper methods that automates manual work*
    - <https://docs.aws.amazon.com/cdk/api/latest/>

## What tooling does AWS CDK offer for containerized applications?

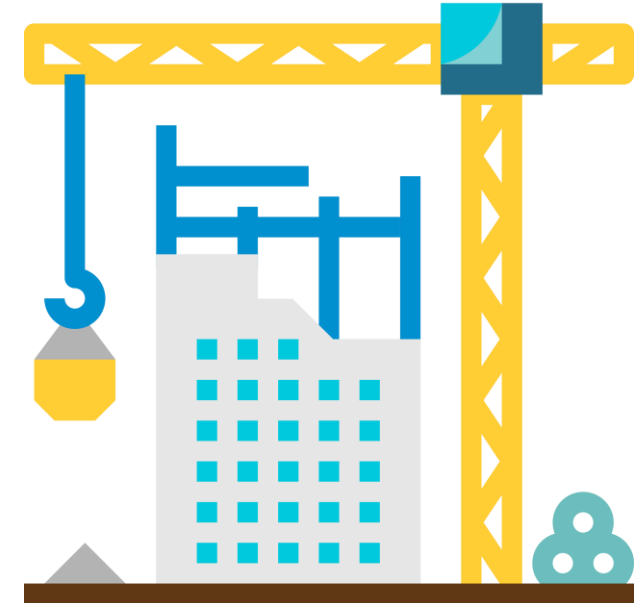
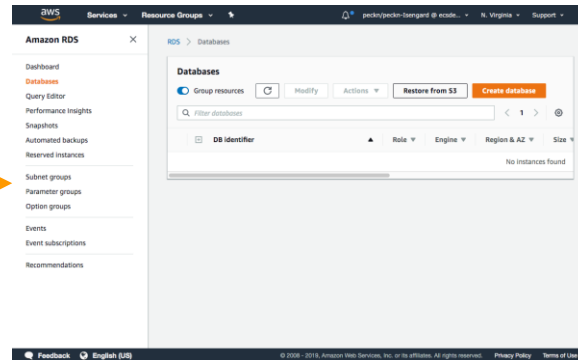
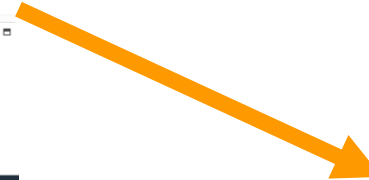
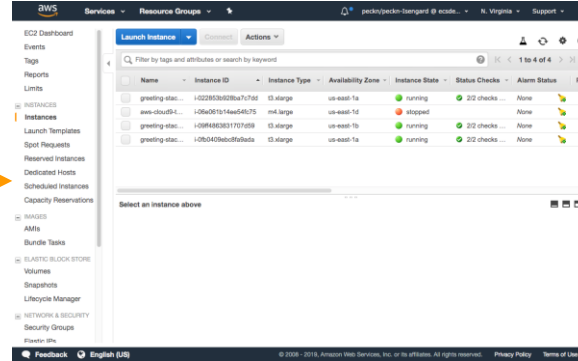
- High level reusable patterns for containers
  - *aws-ecs-patterns (Beginner, High-level), aws-ecs (Advanced, Low-level) constructs*
- Underlying core constructs for more advanced configurations

# Agenda

- IaC levels (ways to implement IaC) and their Pros/Cons
  - Level 0 – IaC by hand
  - Level 1 – Imperative IaC
  - Level 2 – Declarative IaC
  - Level 3 – CDK (Infra is Code, Infra as Class)
- Demos involving containerized applications
  - Deploy a container locally via Docker
  - Deploy Containerized (ECS) Web App with AWS CDK using [aws-ecs-patterns](#)
  - Deploy Containerized (ECS) Web App using [aws-ecs](#) construct
  - Deploy Serverless App (Lambda) with integration to SQS / DynamoDB
- Q&A

# Ways to Implement Infrastructure as Code

# Level 0 – Creating infrastructure by hand



Your organization's infrastructure

# Level 0 – Creating infrastructure by hand



## Pros

- Decent for standing up your first exploratory project infrastructure
- Tight interaction with console can help you see errors faster

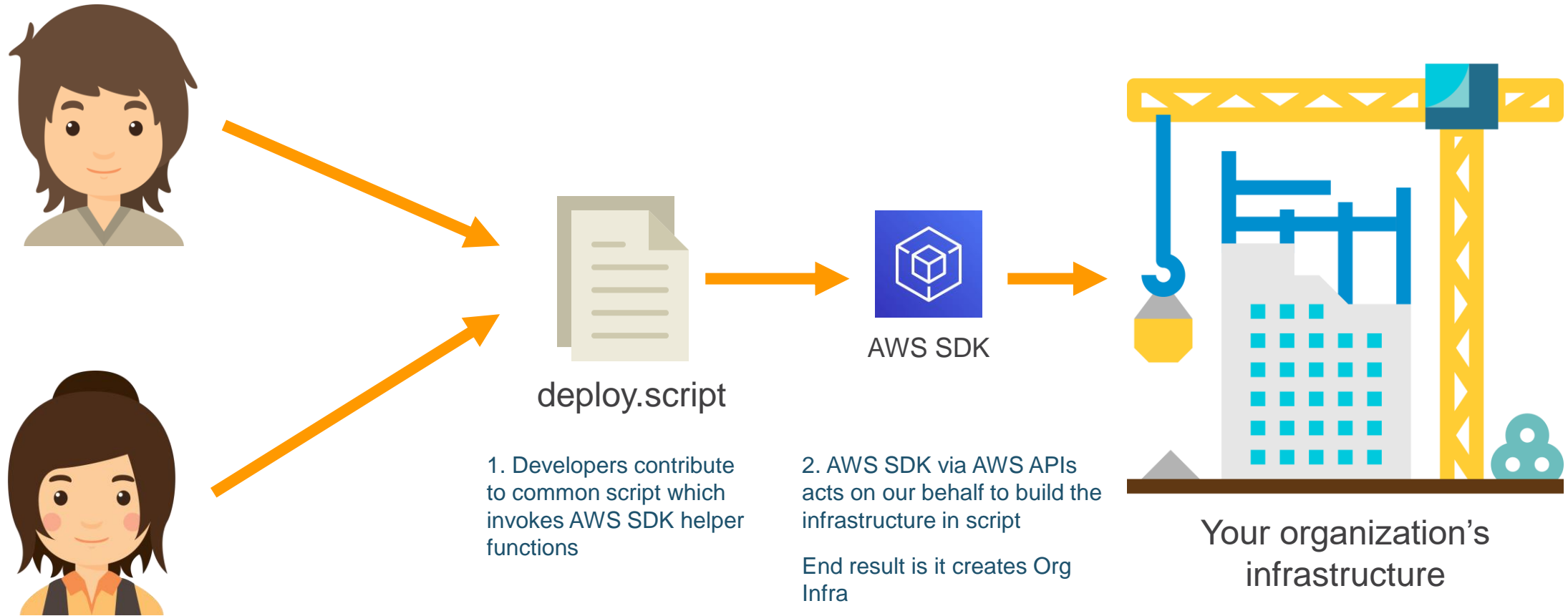


- Clicking things and entering values in the console by hand is **slow**. **Bottleneck** for long-term development
- It's hard to reliably **reproduce** your results when you are doing things manually, by hand.
- People make **mistakes** in data entry and clicking options.  
*Can't standardize reliability*
- Person A configures things one way, but person B configures things another way  
*Can't standardize consistency*



## Cons

# Level 1 – Imperative infrastructure as code



# Level 1 – Imperative infrastructure as code



deploy.script

```
resource = getResource(xyz)

if (resource == desiredResource) {
  return
} else if (!resource) {
  createResource(desiredResource)
} else {
  updateResource(desiredResource)
}
```

- Lots of boilerplate
- What if something fails and we need to retry?
- What if two people try to run the script at once?
- Race conditions?
- Handle edge cases with new requirements. Leads to **bulky, unreliable code**



# Level 1 – Imperative infrastructure as code



## Pros

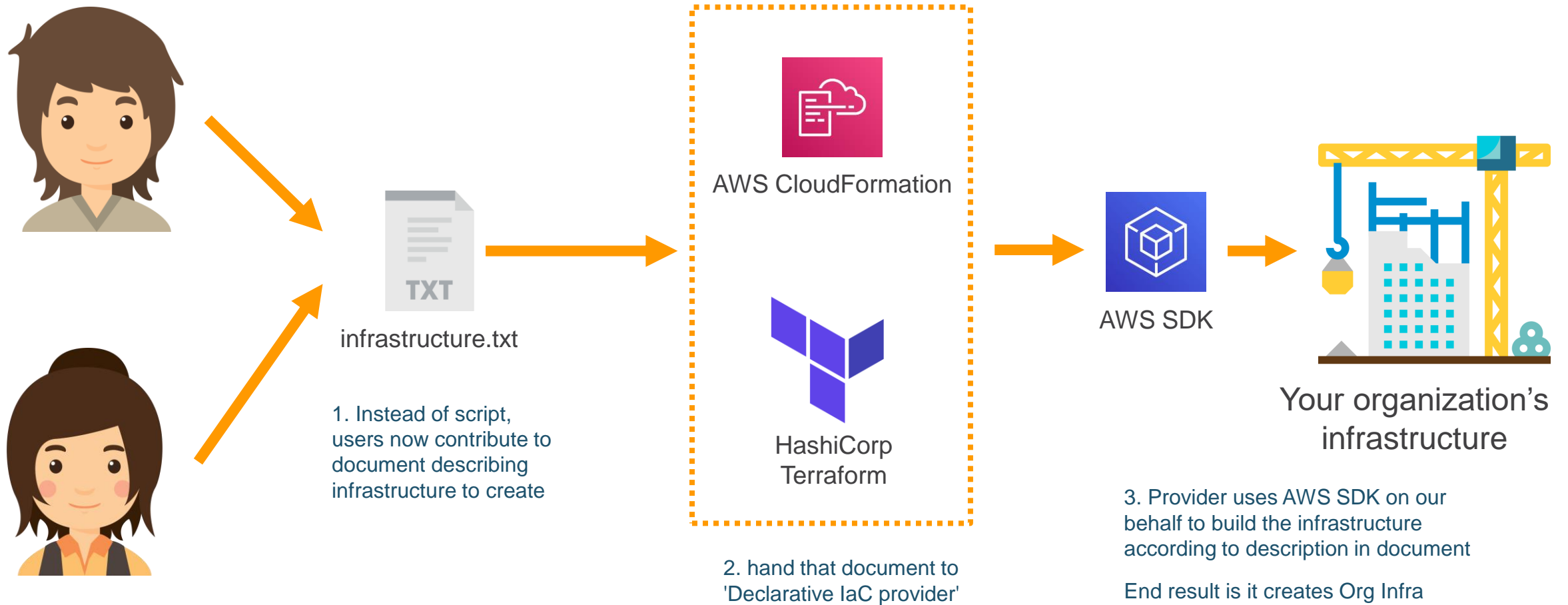
- If the code is written well it is repeatable and reusable
- Multiple people can work on the script collaboratively, fixing bugs, see all the settings in one place

- Lots of boilerplate code to write, and it can be hard to write reliable code
- Imperative code has to **handle** all edge cases
- Must be careful about multiple people using the script at once



## Cons

# Level 2 – Declarative infrastructure as code



# Level 2 – Declarative infrastructure as code

Narrate your Infrastructure resources into the YAML document in a structured meta-data format 😊



infrastructure.txt

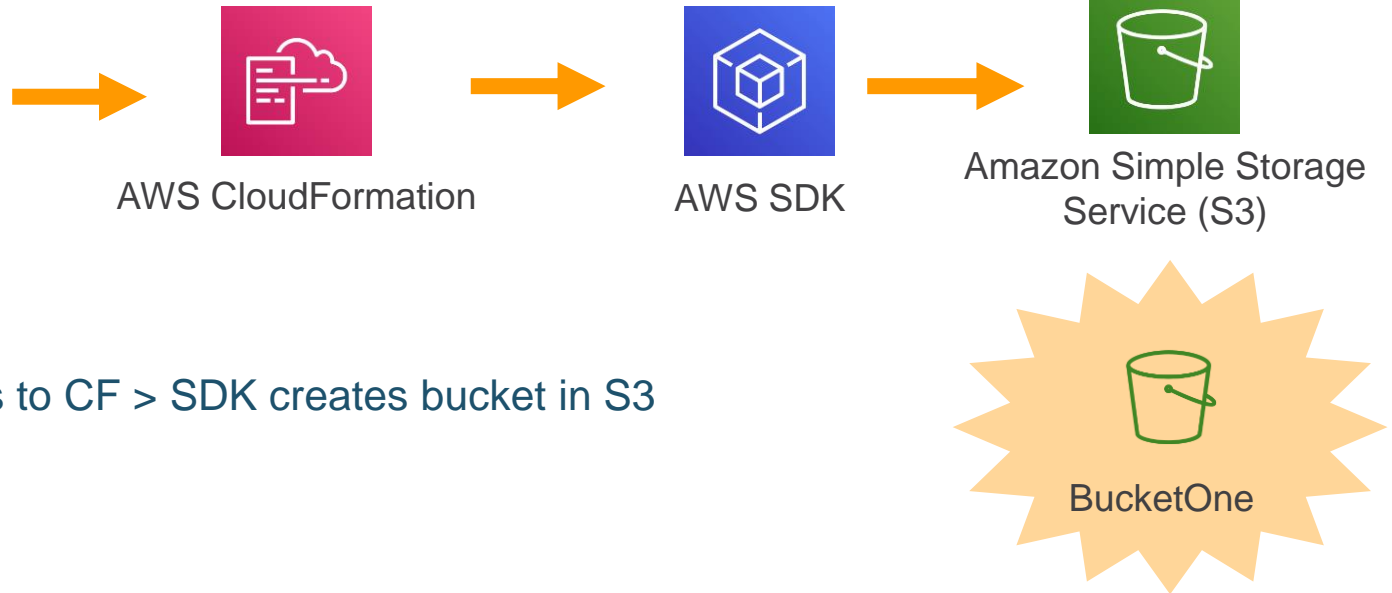
```
Resources:
  # VPC in which containers will be networked.
  # It has two public subnets
  # We distribute the subnets across the first two available subnets
  # for the region, for high availability.
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock: !FindInMap ['SubnetConfig', 'VPC', 'CIDR']

  # Two public subnets, where containers can have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
        Fn::Select:
          - 0
          - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicOne', 'CIDR']
      MapPublicIpOnLaunch: true
  PublicSubnetTwo:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
        Fn::Select:
          - 1
          - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicTwo', 'CIDR']
      MapPublicIpOnLaunch: true
```

- List every resource to create and its properties, in YAML format in this case
- **Helper functions** may be built in to aid in fetching values dynamically.
- Not writing code that runs logic but describe what needs to be created and that translates to logic which gets run

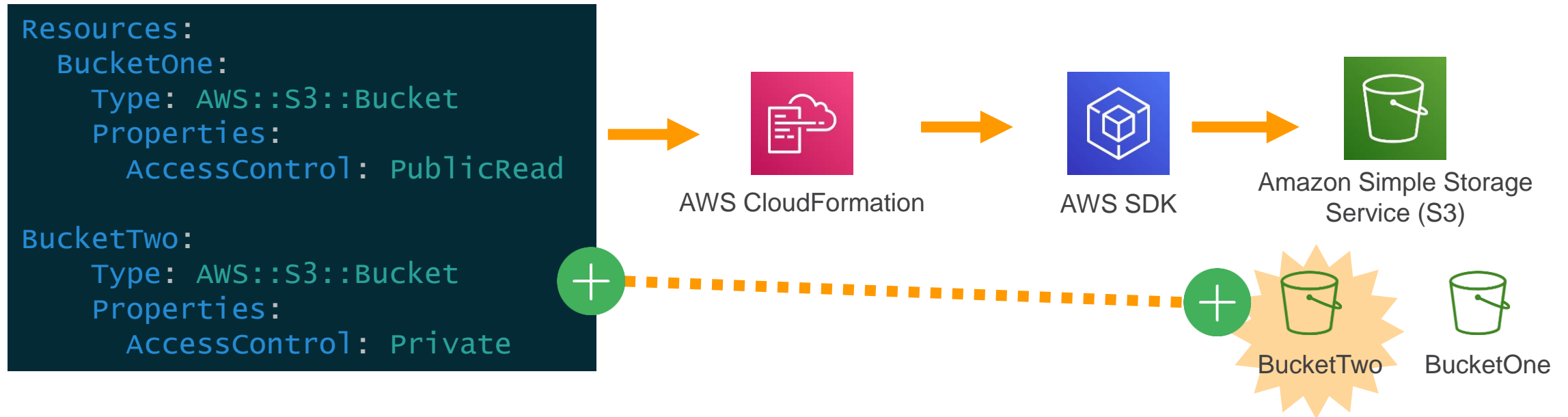
# Level 2 – Declarative infrastructure as code

```
Resources:
  BucketOne:
    Type: AWS::S3::Bucket
    Properties:
      AccessControl: PublicRead
```



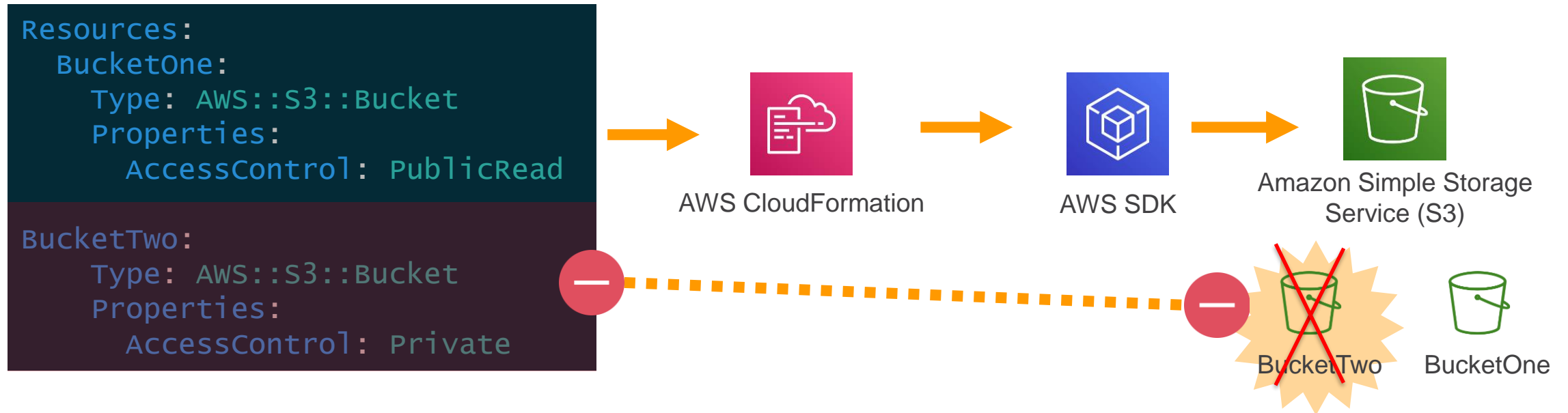
S3 bucket described > Pass to CF > SDK creates bucket in S3

# Level 2 – Declarative infrastructure as code



Add or Delete S3 bucket from doc is translated on the infra to add or delete S3 bucket

# Level 2 – Declarative infrastructure as code



# Level 2 – Declarative infrastructure as code



## Pros

- No imperative boilerplate to write, creating and updating resources is handled **automatically**
- Multiple people can work on the template **collaboratively** since CF locks stack
- Conflict resolution, and resource locking can be handled **centrally**

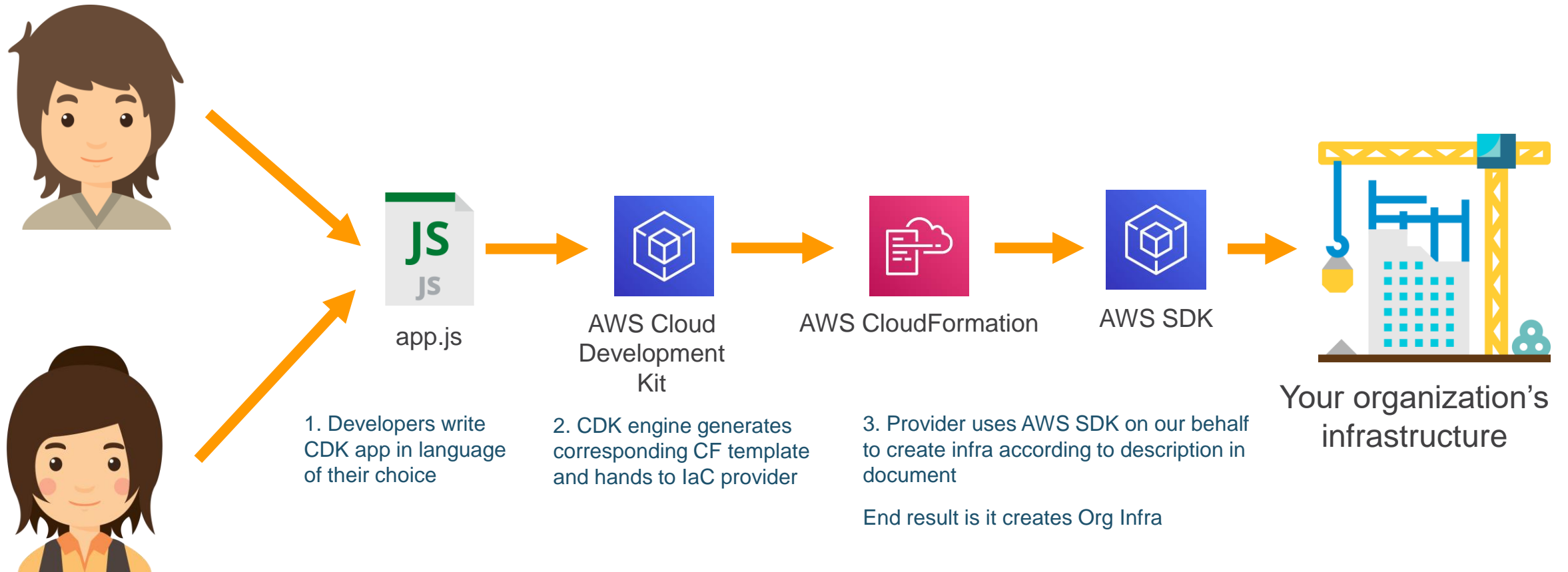


- **1 to 1** relationship between resources in file and resources on account means lots of boilerplate to write. Templates can be very verbose
- Limited ability to run logic as the file formats are generally things like JSON, YAML, or HCL which have only a few built in functions
- Hard to keep things **DRY** (Don't Repeat Yourself) without loops, functions, etc.



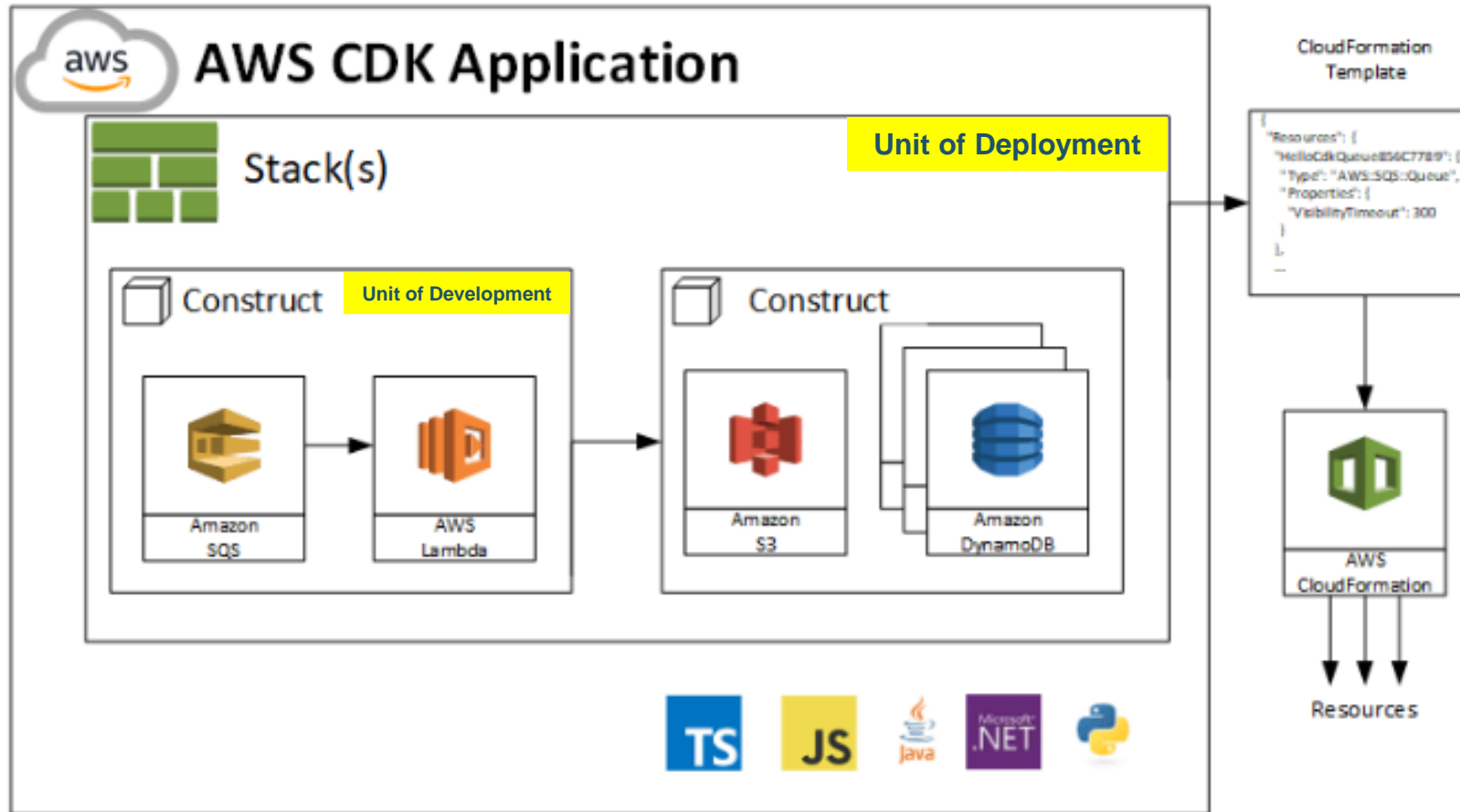
## Cons

# Level 3 – AWS Cloud Development Kit





# Level 3 – AWS Cloud Development Kit



- **App** - A collection of related stacks.
- **Stack** - The set of AWS resources that are created and managed as a single unit when AWS CloudFormation instantiates a template
- **Construct** - Everything defined in the CDK is a **Construct**. It can be thought of as a re-usable "Cloud Component" representing anything from a single AWS resource to architectures of arbitrary complexity.

# Level 3 – AWS Cloud Development Kit



app.js



app.py

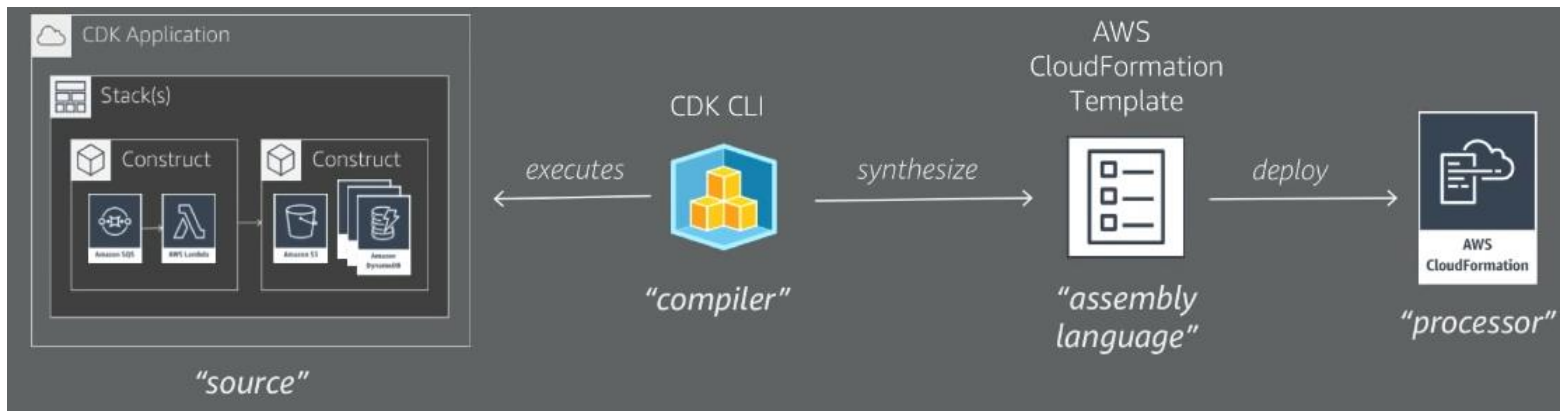
```
class MyService extends cdk.Stack {
  constructor(scope: cdk.App, id: string) {
    super(scope, id);

    // VPC Construct: Network for all the resources
    const vpc = new ec2.Vpc(this, 'MyVpc', { maxAzs: 2 });

    // Cluster Construct: Cluster to hold all the containers
    const cluster = new ecs.Cluster(this, 'Cluster', { vpc: vpc });

    // Load balancer Construct for the service
    const LB = new elbv2.ApplicationLoadBalancer(this, 'LB', {
      vpc: vpc,
      internetFacing: true
    });
  }
}
```

- Write in a familiar programming language
- Each **stack** is made up of “**constructs**” which are simple classes in the code
- Create **many** underlying AWS resources at once with a **single** construct (ec2.Vpc, ecs.Cluster)
- Resources organized into a Stack within same application
- Still declarative, no need to handle create vs update



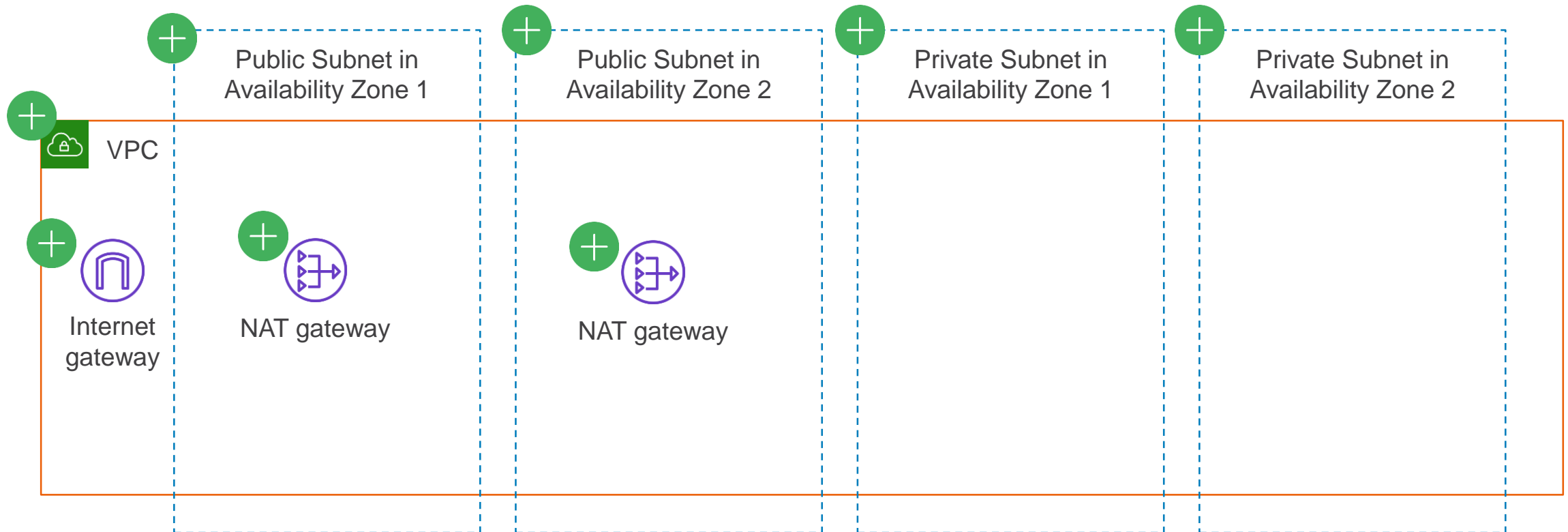
One CDK construct expands to many underlying resources



```
// Network for all the resources
```

```
const vpc = new ec2.vpc(stack, 'MyVpc', { maxAzs: 2 });
```

**cdk deploy**



One CDK construct expands to many underlying resources



```
// Network for all the resources
const vpc = new ec2.Vpc(stack, 'MyVpc', { maxAzs: 2 });
```

cdk synth (-j)

CDK Synth – synthesizes your constructs into CF template(s)



```
1 Resources
2   MyVpcPFC6AF:
3     Type: AWS::EC2::VPC
4     Properties:
5       CidrBlock: 10.0.0.0/16
6       EnableDnsSupport: true
7       EnableDnsHostnames: true
8       InstanceTenancy: default
9     Tags:
10      - Key: Name
11        Value: public-fargate-service/MyVpc
12   Metadata:
13     aws:cdk:path: public-fargate-service/MyVpc/Resource
14   MyVpcPublicSubnet1Subnet668450:
15     Type: AWS::EC2::Subnet
16     Properties:
17       CidrBlock: 10.0.0.0/18
18       VpcId:
19         Ref: MyVpcPFC6AF
20       AvailabilityZone:
21         Fn::Select:
22           - Fn::GetAZs: ""
23       MapPublicIpOnLaunch: true
24     Tags:
25      - Key: Name
26        Value: public-fargate-service/MyVpc/PublicSubnet1
27   Metadata:
28     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet1/Subnet
29   MyVpcPublicSubnet2Subnet668450:
30     Type: AWS::EC2::Subnet
31     Properties:
32       CidrBlock: 10.0.0.0/18
33       VpcId:
34         Ref: MyVpcPFC6AF
35       AvailabilityZone:
36         Fn::Select:
37           - Fn::GetAZs: ""
38       MapPublicIpOnLaunch: true
39     Tags:
40      - Key: Name
41        Value: public-fargate-service/MyVpc/PublicSubnet2
42   Metadata:
43     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet2/Subnet
44   MyVpcPublicSubnet1RouteTableECCECE:
45     Type: AWS::EC2::RouteTable
46     Properties:
47       VpcId:
48         Ref: MyVpcPFC6AF
49       SubnetId:
50         Ref: MyVpcPublicSubnet1Subnet668450
51     Tags:
52      - Key: Name
53        Value: public-fargate-service/MyVpc/PublicSubnet1/RouteTable
54   Metadata:
55     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet1/RouteTable
56   MyVpcPublicSubnet2RouteTableECCECE:
57     Type: AWS::EC2::RouteTable
58     Properties:
59       VpcId:
60         Ref: MyVpcPFC6AF
61       SubnetId:
62         Ref: MyVpcPublicSubnet2Subnet668450
63     Tags:
64      - Key: Name
65        Value: public-fargate-service/MyVpc/PublicSubnet2/RouteTable
66   Metadata:
67     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet2/RouteTable
68   MyVpcPublicSubnet1DefaultRoute959F0B8:
69     Type: AWS::EC2::Route
70     Properties:
71       RouteTableId:
72         Ref: MyVpcPublicSubnet1RouteTableECCECE
73       DestinationCidrBlock: 0.0.0.0/0
74       GatewayId:
75         Ref: MyVpcIGW5CAAF63
76     Tags:
77      - Key: Name
78        Value: public-fargate-service/MyVpc/PublicSubnet1/DefaultRoute
79   Metadata:
80     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet1/DefaultRoute
81   MyVpcPublicSubnet2DefaultRoute959F0B8:
82     Type: AWS::EC2::Route
83     Properties:
84       RouteTableId:
85         Ref: MyVpcPublicSubnet2RouteTableECCECE
86       DestinationCidrBlock: 0.0.0.0/0
87       GatewayId:
88         Ref: MyVpcIGW5CAAF63
89     Tags:
90      - Key: Name
91        Value: public-fargate-service/MyVpc/PublicSubnet2/DefaultRoute
92   Metadata:
93     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet2/DefaultRoute
94   MyVpcPublicSubnet1NATGatewayAD348BC1:
95     Type: AWS::EC2::NATGateway
96     Properties:
97       SubnetId:
98         Ref: MyVpcPublicSubnet1Subnet668450
99       AllocationId:
100        Fn::GetAZs: ""
101       InstanceType:
102         Ref: MyVpcPFC6AF
103       Tags:
104        - Key: Name
105          Value: public-fargate-service/MyVpc/PublicSubnet1/NATGateway
106   Metadata:
107     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet1/NATGateway
108   MyVpcPublicSubnet2NATGateway91F8EC9:
109     Type: AWS::EC2::NATGateway
110     Properties:
111       SubnetId:
112         Ref: MyVpcPublicSubnet2Subnet668450
113       AllocationId:
114        Fn::GetAZs: ""
115       InstanceType:
116         Ref: MyVpcPFC6AF
117       Tags:
118        - Key: Name
119          Value: public-fargate-service/MyVpc/PublicSubnet2/NATGateway
120   Metadata:
121     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet2/NATGateway
122   MyVpcPrivateSubnet1Subnet668450:
123     Type: AWS::EC2::Subnet
124     Properties:
125       CidrBlock: 10.0.0.0/18
126       VpcId:
127         Ref: MyVpcPFC6AF
128       AvailabilityZone:
129         Fn::Select:
130           - Fn::GetAZs: ""
131       MapPublicIpOnLaunch: false
132     Tags:
133      - Key: Name
134        Value: public-fargate-service/MyVpc/PrivateSubnet1
135   Metadata:
136     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/Subnet
137   MyVpcPrivateSubnet2Subnet668450:
138     Type: AWS::EC2::Subnet
139     Properties:
140       CidrBlock: 10.0.0.0/18
141       VpcId:
142         Ref: MyVpcPFC6AF
143       AvailabilityZone:
144         Fn::Select:
145           - Fn::GetAZs: ""
146       MapPublicIpOnLaunch: false
147     Tags:
148      - Key: Name
149        Value: public-fargate-service/MyVpc/PrivateSubnet2
150   Metadata:
151     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/Subnet
152   MyVpcPrivateSubnet1RouteTableECCECE:
153     Type: AWS::EC2::RouteTable
154     Properties:
155       VpcId:
156         Ref: MyVpcPFC6AF
157       SubnetId:
158         Ref: MyVpcPrivateSubnet1Subnet668450
159     Tags:
160      - Key: Name
161        Value: public-fargate-service/MyVpc/PrivateSubnet1/RouteTable
162   Metadata:
163     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/RouteTable
164   MyVpcPrivateSubnet2RouteTableECCECE:
165     Type: AWS::EC2::RouteTable
166     Properties:
167       VpcId:
168         Ref: MyVpcPFC6AF
169       SubnetId:
170         Ref: MyVpcPrivateSubnet2Subnet668450
171     Tags:
172      - Key: Name
173        Value: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
174   Metadata:
175     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
176   MyVpcPrivateSubnet1DefaultRoute959F0B8:
177     Type: AWS::EC2::Route
178     Properties:
179       RouteTableId:
180         Ref: MyVpcPrivateSubnet1RouteTableECCECE
181       DestinationCidrBlock: 0.0.0.0/0
182       GatewayId:
183         Ref: MyVpcIGW5CAAF63
184     Tags:
185      - Key: Name
186        Value: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
187   Metadata:
188     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
189   MyVpcPrivateSubnet2DefaultRoute959F0B8:
190     Type: AWS::EC2::Route
191     Properties:
192       RouteTableId:
193         Ref: MyVpcPrivateSubnet2RouteTableECCECE
194       DestinationCidrBlock: 0.0.0.0/0
195       GatewayId:
196         Ref: MyVpcIGW5CAAF63
197     Tags:
198      - Key: Name
199        Value: public-fargate-service/MyVpc/PrivateSubnet2/DefaultRoute
200   Metadata:
201     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/DefaultRoute
202   MyVpcVPCGW5CAAF63:
203     Type: AWS::EC2::VPCGatewayAttachment
204     Properties:
205       VpcId:
206         Ref: MyVpcPFC6AF
207       SubnetId:
208         Ref: MyVpcPublicSubnet1Subnet668450
209     Tags:
210      - Key: Name
211        Value: public-fargate-service/MyVpc/VPCGW
212   Metadata:
213     aws:cdk:path: public-fargate-service/MyVpc/VPCGW
```

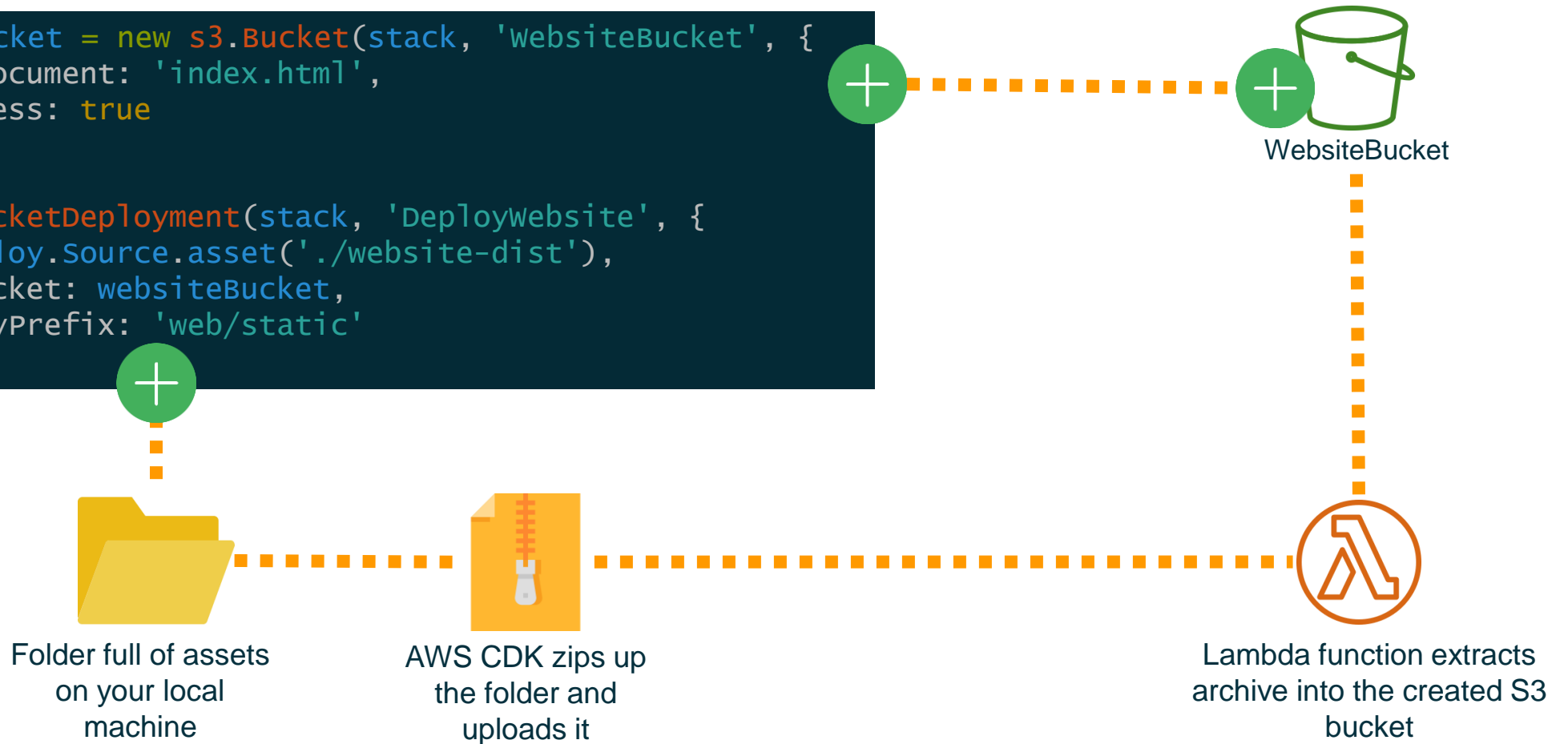
```
181   - Key: aws-cdk:subnet-type
182     Value: Public
183   Metadata:
184     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet1/Subnet
185   MyVpcPublicSubnet2Subnet668450:
186     Type: AWS::EC2::Subnet
187     Properties:
188       CidrBlock: 10.0.0.0/18
189       VpcId:
190         Ref: MyVpcPFC6AF
191       AvailabilityZone:
192         Fn::Select:
193           - Fn::GetAZs: ""
194       MapPublicIpOnLaunch: false
195     Tags:
196      - Key: Name
197        Value: public-fargate-service/MyVpc/PublicSubnet2
198   Metadata:
199     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet2/Subnet
200   MyVpcPublicSubnet1RouteTableECCECE:
201     Type: AWS::EC2::RouteTable
202     Properties:
203       VpcId:
204         Ref: MyVpcPFC6AF
205       SubnetId:
206         Ref: MyVpcPublicSubnet1Subnet668450
207     Tags:
208      - Key: Name
209        Value: public-fargate-service/MyVpc/PublicSubnet1/RouteTable
210   Metadata:
211     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet1/RouteTable
212   MyVpcPublicSubnet2RouteTableECCECE:
213     Type: AWS::EC2::RouteTable
214     Properties:
215       VpcId:
216         Ref: MyVpcPFC6AF
217       SubnetId:
218         Ref: MyVpcPublicSubnet2Subnet668450
219     Tags:
220      - Key: Name
221        Value: public-fargate-service/MyVpc/PublicSubnet2/RouteTable
222   Metadata:
223     aws:cdk:path: public-fargate-service/MyVpc/PublicSubnet2/RouteTable
224   MyVpcPrivateSubnet1Subnet668450:
225     Type: AWS::EC2::Subnet
226     Properties:
227       CidrBlock: 10.0.0.0/18
228       VpcId:
229         Ref: MyVpcPFC6AF
230       AvailabilityZone:
231         Fn::Select:
232           - Fn::GetAZs: ""
233       MapPublicIpOnLaunch: false
234     Tags:
235      - Key: Name
236        Value: public-fargate-service/MyVpc/PrivateSubnet1
237   Metadata:
238     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/Subnet
239   MyVpcPrivateSubnet2Subnet668450:
240     Type: AWS::EC2::Subnet
241     Properties:
242       CidrBlock: 10.0.0.0/18
243       VpcId:
244         Ref: MyVpcPFC6AF
245       AvailabilityZone:
246         Fn::Select:
247           - Fn::GetAZs: ""
248       MapPublicIpOnLaunch: false
249     Tags:
250      - Key: Name
251        Value: public-fargate-service/MyVpc/PrivateSubnet2
252   Metadata:
253     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/Subnet
254   MyVpcPrivateSubnet1RouteTableECCECE:
255     Type: AWS::EC2::RouteTable
256     Properties:
257       VpcId:
258         Ref: MyVpcPFC6AF
259       SubnetId:
260         Ref: MyVpcPrivateSubnet1Subnet668450
261     Tags:
262      - Key: Name
263        Value: public-fargate-service/MyVpc/PrivateSubnet1/RouteTable
264   Metadata:
265     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/RouteTable
266   MyVpcPrivateSubnet2RouteTableECCECE:
267     Type: AWS::EC2::RouteTable
268     Properties:
269       VpcId:
270         Ref: MyVpcPFC6AF
271       SubnetId:
272         Ref: MyVpcPrivateSubnet2Subnet668450
273     Tags:
274      - Key: Name
275        Value: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
276   Metadata:
277     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
278   MyVpcPrivateSubnet1DefaultRoute959F0B8:
279     Type: AWS::EC2::Route
280     Properties:
281       RouteTableId:
282         Ref: MyVpcPrivateSubnet1RouteTableECCECE
283       DestinationCidrBlock: 0.0.0.0/0
284       GatewayId:
285         Ref: MyVpcIGW5CAAF63
286     Tags:
287      - Key: Name
288        Value: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
289   Metadata:
290     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
291   MyVpcPrivateSubnet2DefaultRoute959F0B8:
292     Type: AWS::EC2::Route
293     Properties:
294       RouteTableId:
295         Ref: MyVpcPrivateSubnet2RouteTableECCECE
296       DestinationCidrBlock: 0.0.0.0/0
297       GatewayId:
298         Ref: MyVpcIGW5CAAF63
299     Tags:
300      - Key: Name
301        Value: public-fargate-service/MyVpc/PrivateSubnet2/DefaultRoute
302   Metadata:
303     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/DefaultRoute
304   MyVpcVPCGW5CAAF63:
305     Type: AWS::EC2::VPCGatewayAttachment
306     Properties:
307       VpcId:
308         Ref: MyVpcPFC6AF
309       SubnetId:
310         Ref: MyVpcPublicSubnet1Subnet668450
311     Tags:
312      - Key: Name
313        Value: public-fargate-service/MyVpc/VPCGW
314   Metadata:
315     aws:cdk:path: public-fargate-service/MyVpc/VPCGW
```

```
200   DestinationCidrBlock: 0.0.0.0/0
201   NatGatewayId:
202     Ref: MyVpcPublicSubnet1NATGatewayAD348BC1
203   Metadata:
204     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
205   MyVpcPrivateSubnet2Subnet668450:
206     Type: AWS::EC2::Subnet
207     Properties:
208       CidrBlock: 10.0.0.0/18
209       VpcId:
210         Ref: MyVpcPFC6AF
211       AvailabilityZone:
212         Fn::Select:
213           - Fn::GetAZs: ""
214       MapPublicIpOnLaunch: false
215     Tags:
216      - Key: Name
217        Value: public-fargate-service/MyVpc/PrivateSubnet2
218   Metadata:
219     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/Subnet
220   MyVpcPrivateSubnet2RouteTableECCECE:
221     Type: AWS::EC2::RouteTable
222     Properties:
223       VpcId:
224         Ref: MyVpcPFC6AF
225       SubnetId:
226         Ref: MyVpcPrivateSubnet2Subnet668450
227     Tags:
228      - Key: Name
229        Value: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
230   Metadata:
231     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
232   MyVpcPrivateSubnet1DefaultRoute959F0B8:
233     Type: AWS::EC2::Route
234     Properties:
235       RouteTableId:
236         Ref: MyVpcPrivateSubnet1RouteTableECCECE
237       DestinationCidrBlock: 0.0.0.0/0
238       GatewayId:
239         Ref: MyVpcIGW5CAAF63
240     Tags:
241      - Key: Name
242        Value: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
243   Metadata:
244     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/DefaultRoute
245   MyVpcPrivateSubnet2DefaultRoute959F0B8:
246     Type: AWS::EC2::Route
247     Properties:
248       RouteTableId:
249         Ref: MyVpcPrivateSubnet2RouteTableECCECE
250       DestinationCidrBlock: 0.0.0.0/0
251       GatewayId:
252         Ref: MyVpcIGW5CAAF63
253     Tags:
254      - Key: Name
255        Value: public-fargate-service/MyVpc/PrivateSubnet2/DefaultRoute
256   Metadata:
257     aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/DefaultRoute
258   MyVpcVPCGW5CAAF63:
259     Type: AWS::EC2::VPCGatewayAttachment
260     Properties:
261       VpcId:
262         Ref: MyVpcPFC6AF
263       SubnetId:
264         Ref: MyVpcPublicSubnet1Subnet668450
265     Tags:
266      - Key: Name
267        Value: public-fargate-service/MyVpc/VPCGW
268   Metadata:
269     aws:cdk:path: public-fargate-service/MyVpc/VPCGW
```

270 lines of  
CloudFormation  
YAML you don't  
have to write!

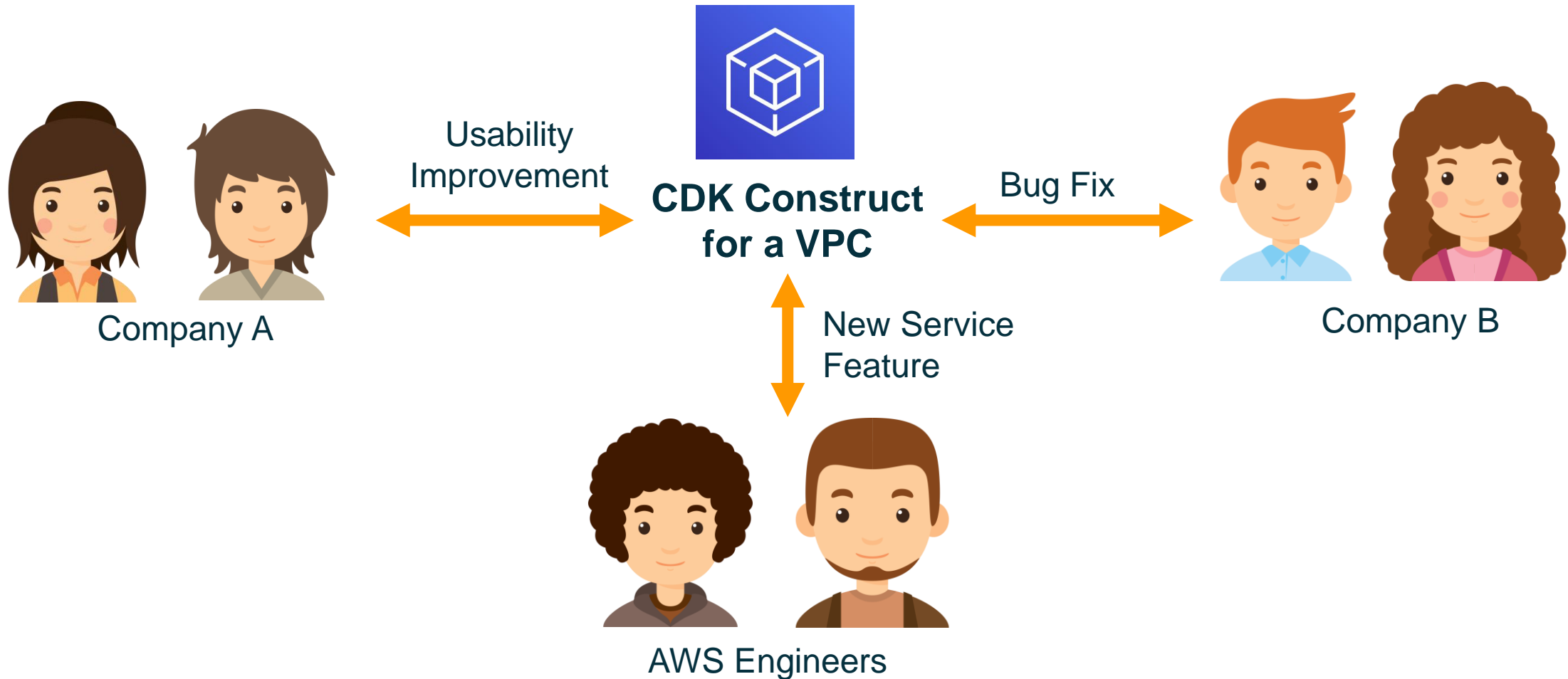
# CDK helps with your local workflow too

```
const websiteBucket = new s3.Bucket(stack, 'WebsiteBucket', {  
  websiteIndexDocument: 'index.html',  
  publicReadAccess: true  
});  
  
new s3deploy.BucketDeployment(stack, 'DeployWebsite', {  
  source: s3deploy.Source.asset('./website-dist'),  
  destinationBucket: websiteBucket,  
  destinationKeyPrefix: 'web/static'  
});
```



# CDK constructs are shareable and reusable

Subscribe to [aws-cdk-interest@amazon.com](mailto:aws-cdk-interest@amazon.com)



# Lots of open source constructs on



alexask  
app-delivery  
assets  
aws-amazonmq  
aws-amplify  
aws-apigateway  
aws-applicationautoscaling  
aws-appmesh  
aws-appstream  
aws-appsync  
aws-athena  
aws-autoscaling  
aws-autoscaling-common  
aws-autoscaling-hooktargets  
aws-autoscalingplans  
aws-backup  
aws-batch

aws-budgets  
aws-certificatemanager  
aws-cloud9  
aws-cloudformation  
aws-cloudfront  
aws-cloudtrail  
aws-cloudwatch  
aws-cloudwatch-actions  
aws-codebuild  
aws-codecommit  
aws-codedeploy  
aws-codepipeline  
aws-codepipeline-actions  
aws-codestar  
aws-cognito  
aws-config  
aws-datapipeline

aws-dax  
aws-directoryservice  
aws-dlm  
aws-dms  
aws-docdb  
aws-dynamodb  
aws-dynamodb-global  
aws-ec2  
aws-ecr  
aws-ecr-assets  
aws-ecs  
aws-ecs-patterns  
aws-efs  
aws-eks  
aws-elasticache  
aws-elasticbeanstalk  
aws-elasticloadbalancing

aws-elasticloadbalancingv2  
aws-elasticloadbalancingv2-targets  
aws-elasticsearch  
aws-emr  
aws-events  
aws-events-targets  
aws-fsx  
aws-gamelift  
aws-glue  
aws-greengrass  
aws-guardduty  
aws-iam  
aws-inspector  
aws-iot  
aws-iotclick  
aws-iotanalytics  
aws-iotevents ... and many more!

# Level 3 – AWS Cloud Development Kit



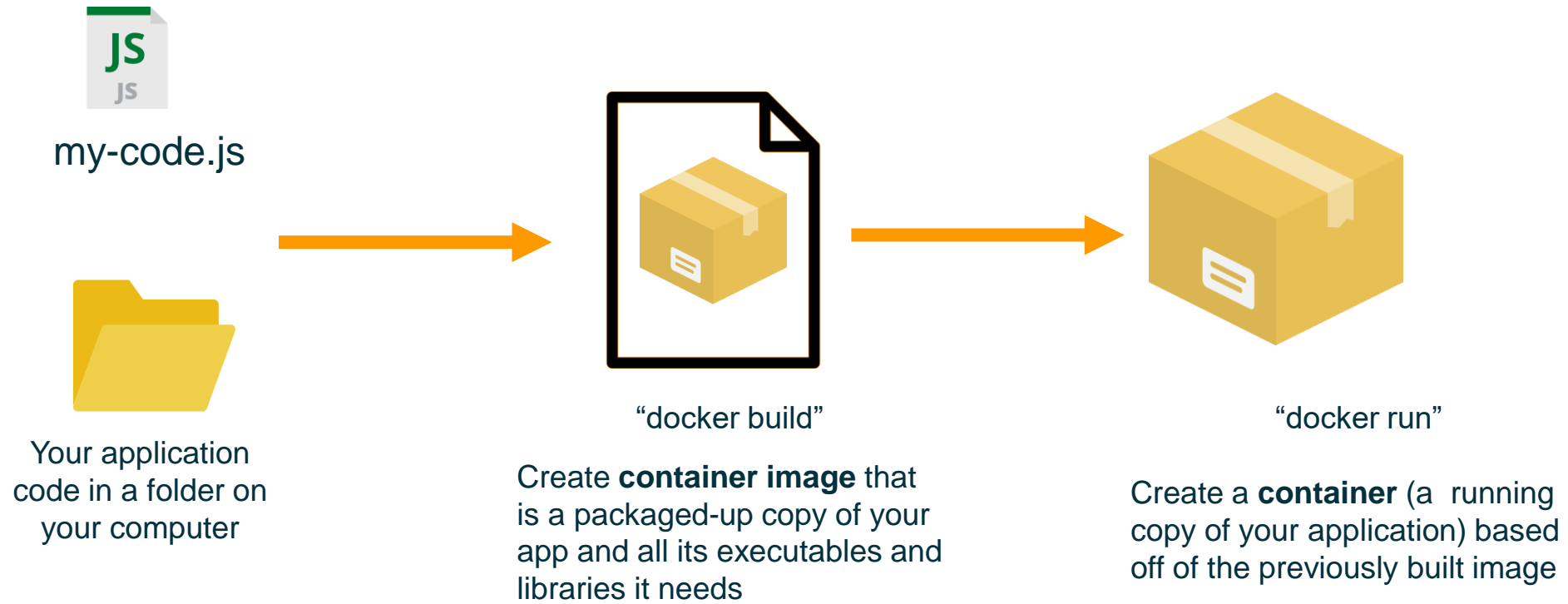
## Pros

- Declarative: creating and updating resources is handled automatically
- Higher level constructs that automatically create many underlying resources
- Multiple people can work on the CDK app collaboratively
- Conflict resolution, and resource locking can be handled centrally
- Use familiar programming languages: Python, JavaScript, TypeScript, .Net, Java
- CDK does more than just create cloud resources, it also helps with your local development workflow
- Easily share and reuse constructs on NPM. Benefit from best practice constructs designed by experts



# AWS CDK for containerized applications

# First some basic container concepts



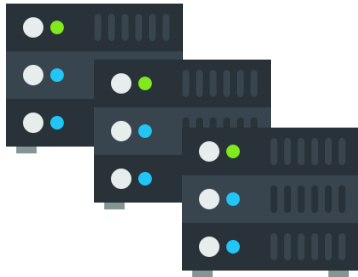
# Add some container orchestration concepts

To launch a containerized application in AWS with ECS, follow the below steps



## 1. Registering task definition

Description of what containerized app to run and what settings it needs e.g. CPU, Memory



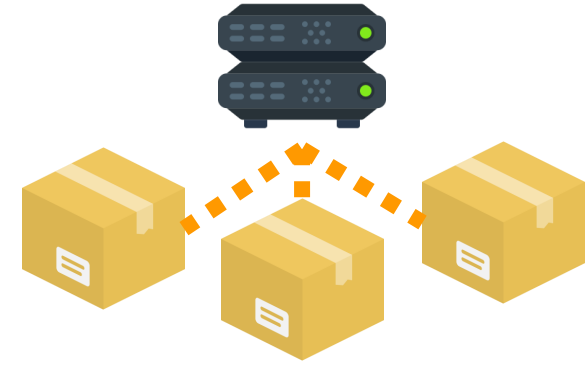
## 2. Create cluster

Capacity for running application, either [EC2 instances](#) or stay serverless with [AWS Fargate](#)



## 3. Run single task (Non-HA)

Launch a standalone task in a cluster based on a task definition description. Just runs until completion then exits



## 4. Create service (HA)

Run multiple copies of a task. Hook them up to other resources like a load balancer. Keep running them until I say to stop

# Approaches to using containers in CDK

## @aws-cdk/aws-ecs-patterns

```
const myService = new ecs_patterns.LoadBalancedFargateService(stack, "my-service", {
  cluster,
  desiredCount: 3,
  image: ecs.ContainerImage.fromAsset("apps/myapp")
});
```

Common architecture patterns built on top of the basic patterns: a **load balanced service**, a queue consumer, task scheduled to run at a particular time.

- If you are **just starting out** with containers recommend using the **ECS patterns** as it is easier to get started with.
- If you are an **experienced ECS user** and want to be able to customize all the settings you normally use then stick to the mid level ECS constructs
- Either way both levels of abstraction remove a lot of boilerplate!

## @aws-cdk/aws-ecs

```
const taskDefinition = new ecs.Ec2TaskDefinition(stack, 'TaskDef');
const container = taskDefinition.addContainer('web', {
  image: ecs.ContainerImage.fromRegistry("apps/myapp"),
  memoryLimitMiB: 256,
});

container.addPortMappings({
  containerPort: 80,
  hostPort: 8080,
  protocol: ecs.Protocol.TCP
});

const service = new ecs.Ec2Service(stack, "Service", {
  cluster,
  taskDefinition,
});

const service = new ecs.Ec2Service(stack, "Service", {
  cluster,
  taskDefinition,
});

const lb = new elbv2.ApplicationLoadBalancer(stack, 'LB', {
  vpc,
  internetFacing: true
});

const listener = lb.addListener('PublicListener', { port: 80, open: true });

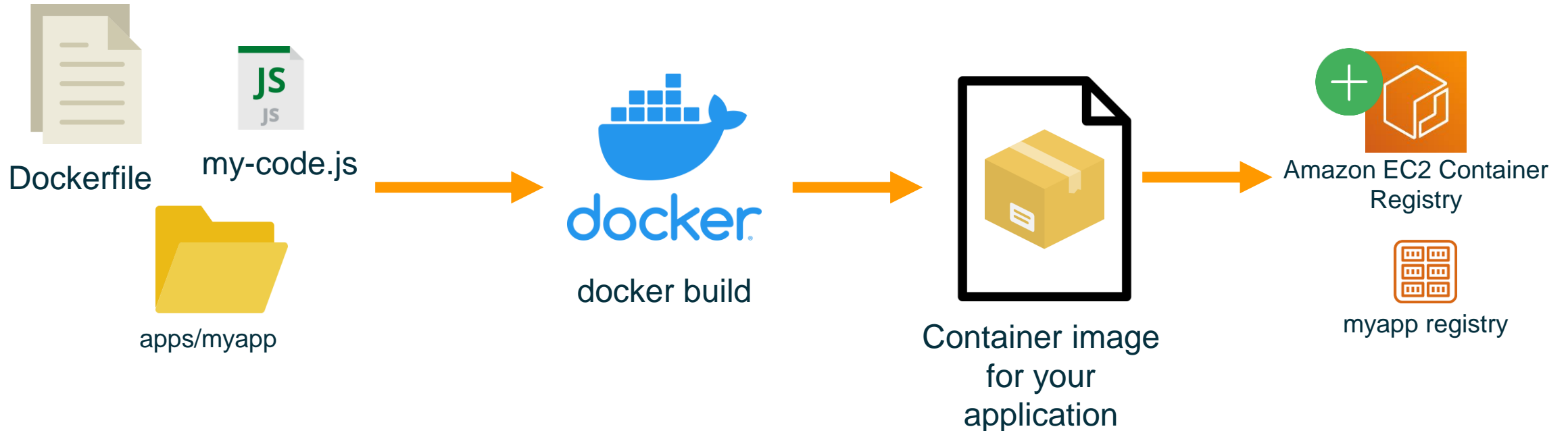
listener.addTarget('ECS', {
  port: 80,
  targets: [service],
  // include health check (default is none)
  healthCheck: {
    interval: cdk.Duration.seconds(60),
    path: "/health",
    timeout: cdk.Duration.seconds(5),
  }
});
```

Basic patterns for building Docker images, creating a cluster, task definition, task, or service.

# @aws-cdk/aws-ecs: Build a container image

This is another example of CDK helping with local workflow, by building and pushing container image to cloud

```
import ecs = require('@aws-cdk/aws-ecs');  
  
const image = ecs.ContainerImage.fromAsset("apps/myapp")
```



*Construct 'fromAsset': Builds the docker image using Dockerfile, creates registry in ECR and pushes it to cloud*

# @aws-cdk/aws-ecs: Create cluster for application

Do you want to stay serverless (Fargate) as below ?

```
import ec2 = require('@aws-cdk/aws-ec2');
import ecs = require('@aws-cdk/aws-ecs');

const vpc = new ec2.Vpc(stack, 'MyVpc', { maxAzs: 2 });
const cluster = new ecs.Cluster(stack, 'Cluster', { vpc });
```

Or do you want to add EC2 instances and run on EC2 as below?

```
cluster.addCapacity('cluster-capacity', {
  instanceType: new ec2.InstanceType("t2.xlarge"),
  desiredCapacity: 3
});
```

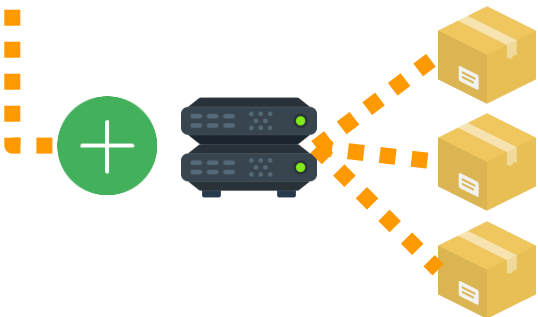
# @aws-cdk/aws-ecs-patterns: Load balanced service

```
import ec2 = require('@aws-cdk/aws-ec2');
import ecs = require('@aws-cdk/aws-ecs');
import ecs_patterns = require('@aws-cdk/aws-ecs-patterns');

const vpc = new ec2.Vpc(stack, 'MyVpc', { maxAzs: 2 });
const cluster = new ecs.Cluster(stack, 'Cluster', { vpc });

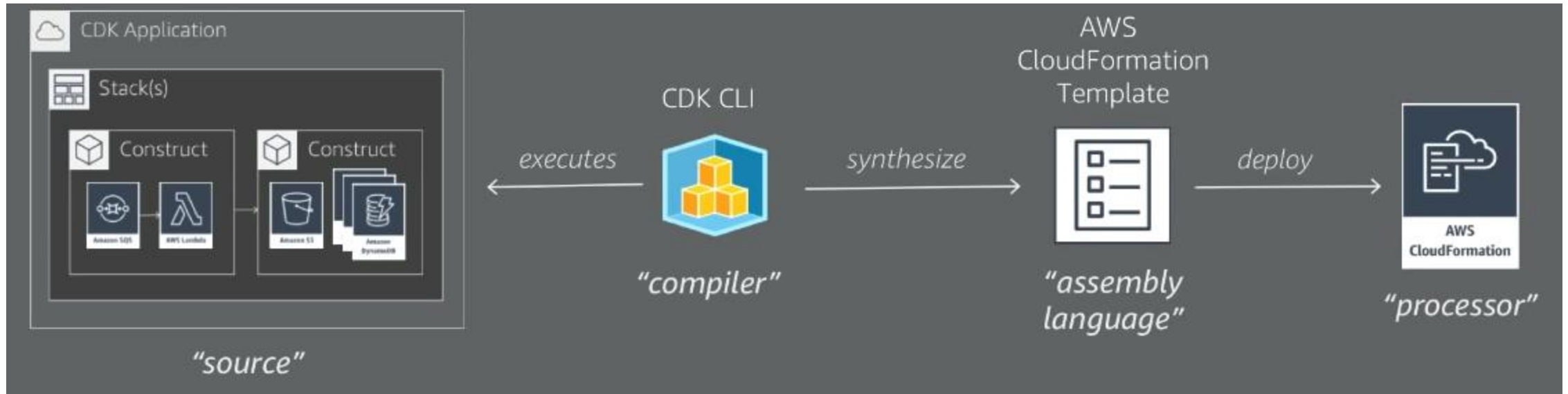
const myService = new ecs_patterns.ApplicationLoadBalancedFargateService(stack, "my-service", {
  cluster,
  desiredCount: 3,
  image: ecs.ContainerImage.fromAsset("apps/myapp")
});
```

**desiredCount=3 is total containers to be run within ECS cluster**



With a few lines we are automatically **building** a Docker container locally, **pushing** it up to the cloud in an Amazon Elastic Container Registry, then **launching** running three copies of it in AWS Fargate, **behind** a load balancer that distributes traffic across all three.

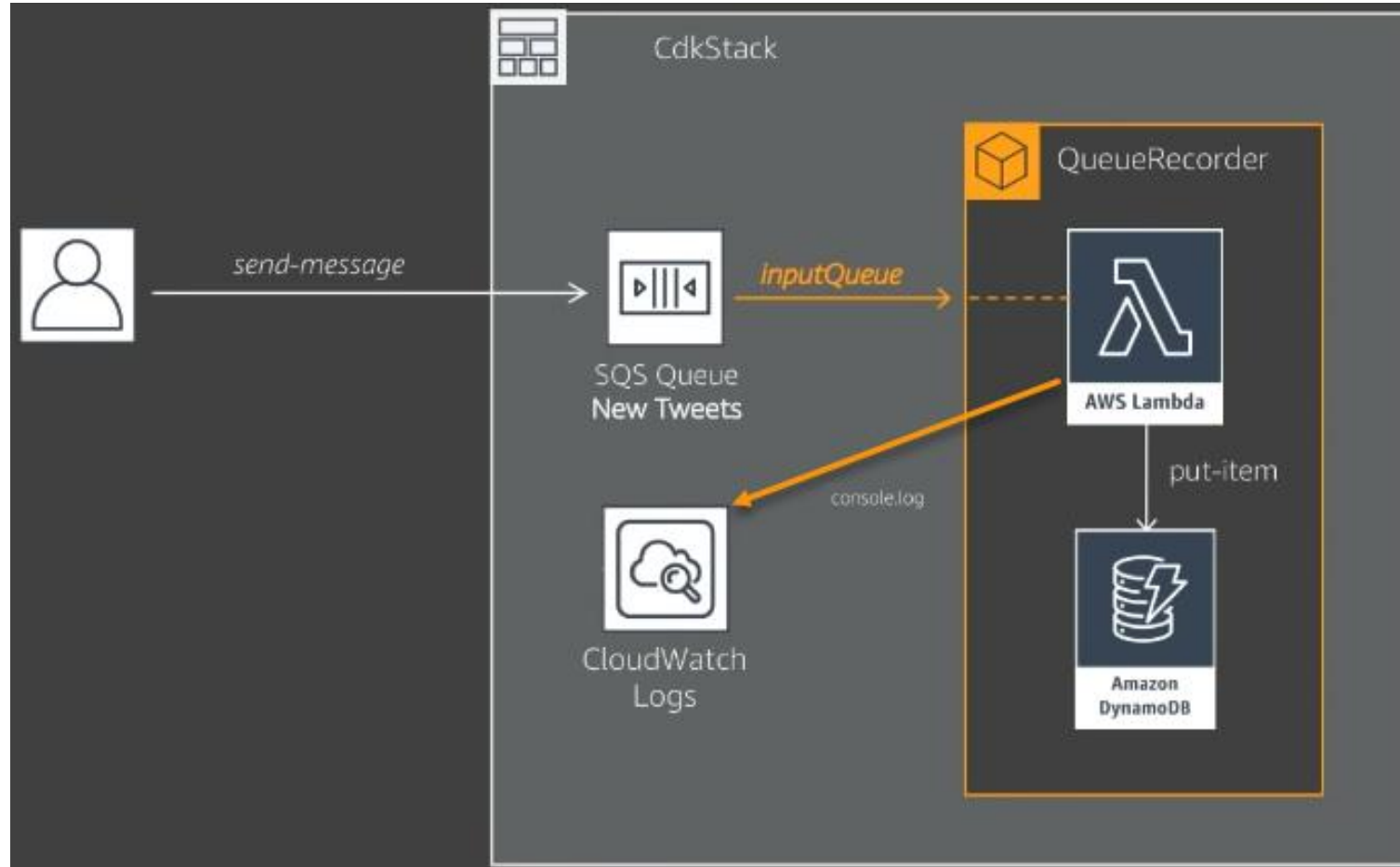
# Deploy containerized (ECS) Web App with AWS CDK



## Demo time!



# Deploy Serverless App with integration to SQS / DynamoDB



## Demo time!

Let's dive a little deeper now

# Create a service manually

```
const taskDefinition = new ecs.Ec2TaskDefinition(stack, 'TaskDef');
const container = taskDefinition.addContainer('web', {
  image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
  memoryLimitMiB: 256,
});

container.addPortMappings({
  containerPort: 80,
  hostPort: 8080,
  protocol: ecs.Protocol.TCP
});

// Create Service
const service = new ecs.Ec2Service(stack, "Service", {
  cluster,
  taskDefinition,
});
```

# Expose service via a load balancer

```
// Create Service
const service = new ecs.Ec2Service(stack, "Service", {
  cluster,
  taskDefinition,
});

// Create ALB
const lb = new elbv2.ApplicationLoadBalancer(stack, 'LB', {
  vpc,
  internetFacing: true
});
const listener = lb.addListener('PublicListener', { port: 80, open: true });

// Attach ALB to ECS Service
listener.addTargetGroups('ECS', {
  port: 80,
  targets: [service],
  // include health check (default is none)
  healthCheck: {
    interval: cdk.Duration.seconds(60),
    path: "/health",
    timeout: cdk.Duration.seconds(5),
  }
});
```

# Add access to some other resources

```
const taskDefinition = new ecs.Ec2TaskDefinition(stack, 'TaskDef');
const container = taskDefinition.addContainer('web', {
  image: ecs.ContainerImage.fromRegistry("apps/myapp"),
  memoryLimitMiB: 256,
});

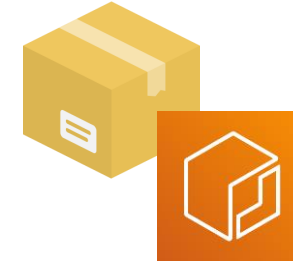
// Grant this task role access to use other resources
myDynamodbTable.grantReadWriteData(taskDefinition.taskRole);
mySnsTopic.grantPublish(taskDefinition.taskRole);
```

- No need to handwrite an IAM policy for your application. CDK already has sensible default access rules built in, and you can grant them to your container applications

# Things AWS CDK can automate away for you



- AWS CDK automatically **creates security groups** and minimal security group rules that allow the load balancer to talk to your tasks



Amazon Elastic Container Registry

- AWS CDK can automatically **build your container image and automatically push** it to an automatically created ECR registry



AWS Identity and Access Management (IAM)



Role

- AWS CDK automatically **creates an IAM role for my task**. You can then easily add minimal access to other resources on my account



Application Load Balancer

- AWS CDK can automatically **create a load balancer** and attach it to your service for you

# **“Deploy Web App using aws-ecs construct”**

## **Demo time!**

# Next steps

- CDK Workshop: <https://cdkworkshop.com/>
- CDK Documentation: <https://docs.aws.amazon.com/cdk/api/latest/>
- Github repo (search for CDK): <https://aws.github.io/>
- Github CDK Samples: <https://github.com/aws-samples/aws-cdk-examples>
- CDK Wiki: <https://w.amazon.com/index.php/AWS/DeveloperResources/AWSSDKsAndTools/CDK>
- Use course code to download and test: <https://github.com/jvargh/aws-cdk-workshop>



# Thanks a lot!

## Q & A