

Deploying Containerized Applications using the Cloud Development Kit (CDK)

November 25, 2019

Joji Varghese



Quick Facts

Introduction to Infrastructure as Code (IaC)

- What is IaC? Why use it?
 - *Concept 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?

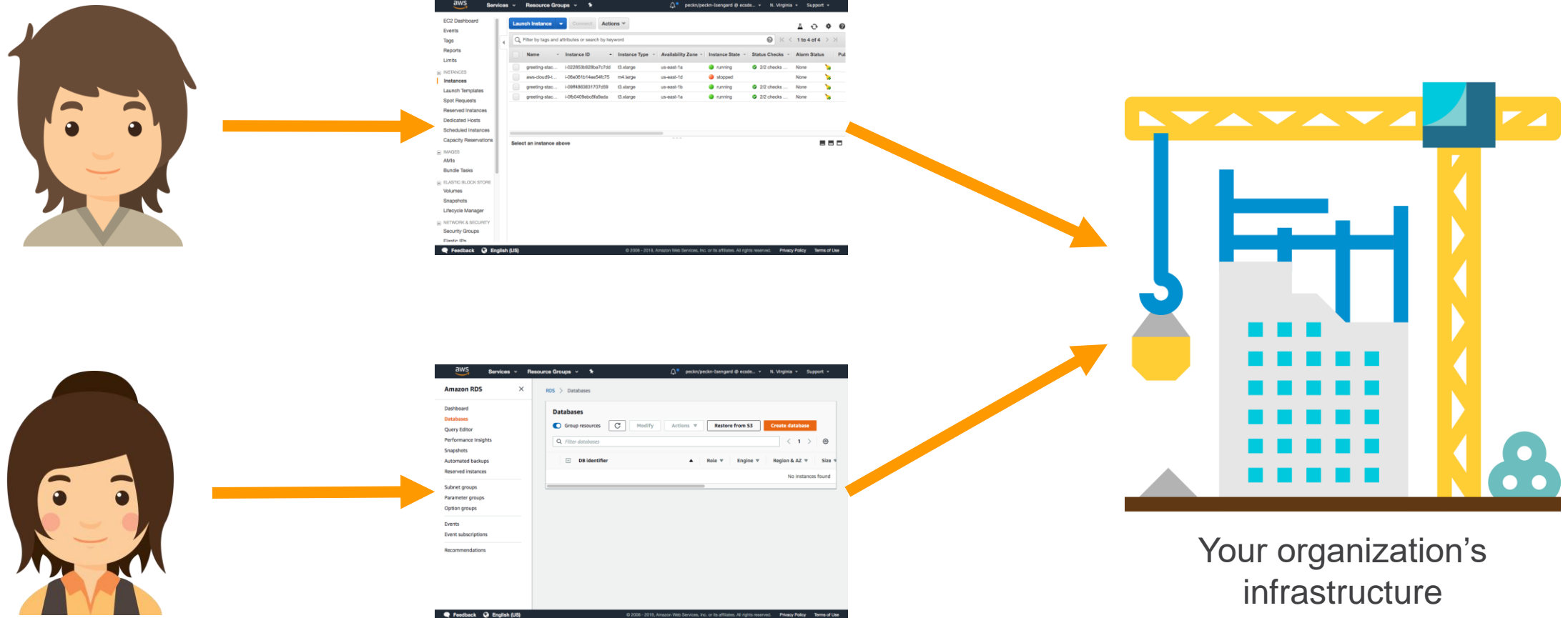
- High level reusable patterns for containers
 - *aws-ecs-patterns (Beginner, High-level), aws-ecs (Advanced, Low-level) constructs*
- Underlying core constructs for more advanced configurations

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](#) pattern
 - Deploy Serverless App (Lambda) with integration to SQS / DynamoDB
- Q&A

Ways to Implement Infrastructure as Code

Level 0 – Creating infrastructure by hand



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

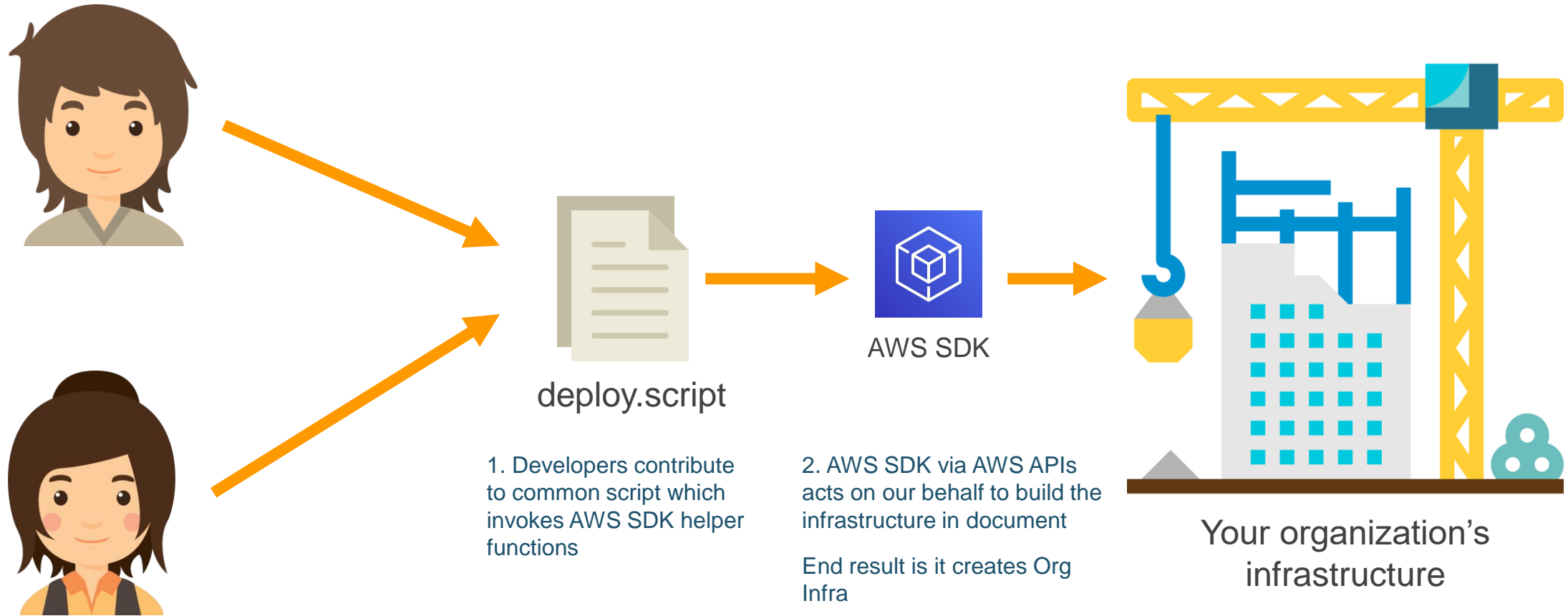


- Clicking things and entering values in the console by hand is **slow**. **Bottleneck** for long-term development
- It's hard to reliably **reproduce** your results when you are doing things manually, by hand.
- People make **mistakes** in data entry and clicking options.
Can't standardize reliability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

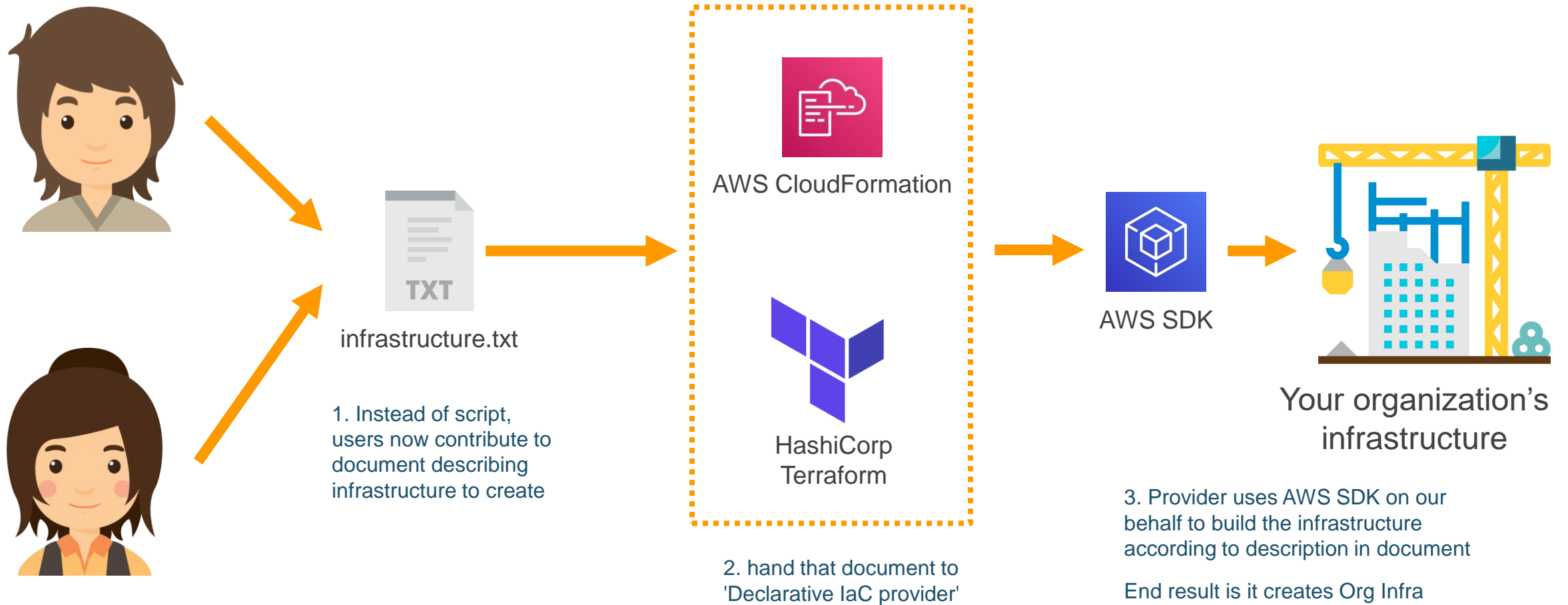
- If the code is written well it is repeatable and reusable
- Multiple people can work on the script collaboratively, fixing bugs, see all the settings in one place

- 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

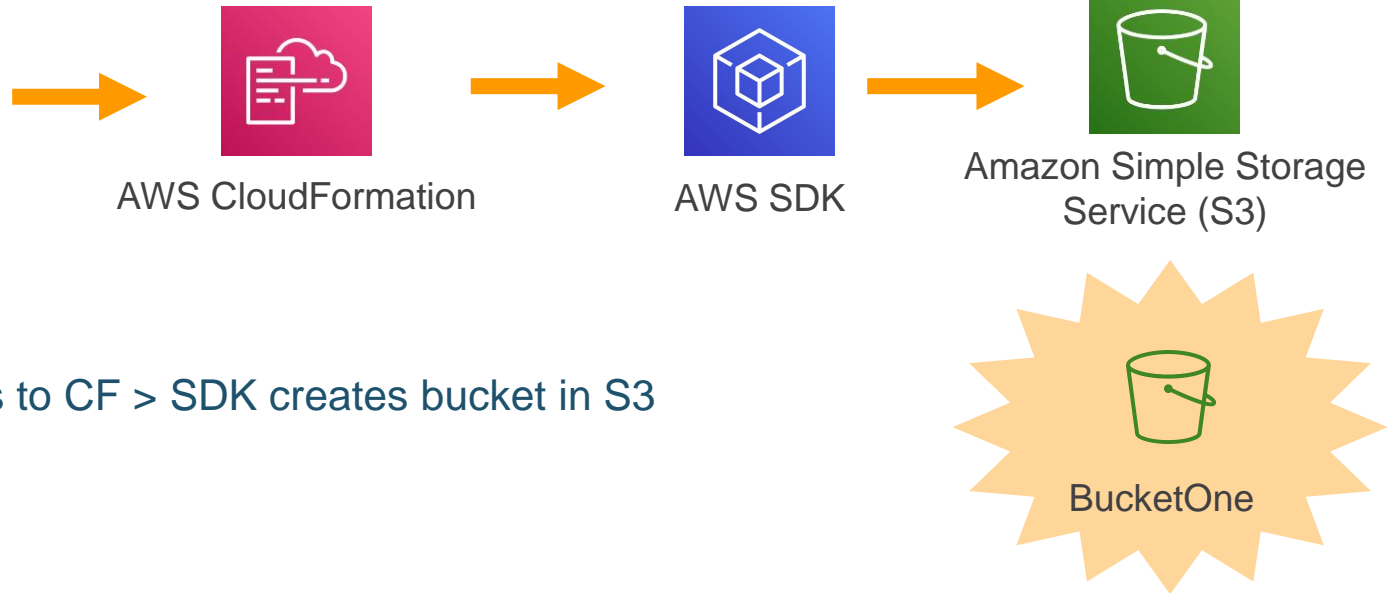
```
Resources:
  # VPC in which containers will be networked.
  # It has two public subnets
  # We distribute the subnets across the first two available subnets
  # for the region, for high availability.
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: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

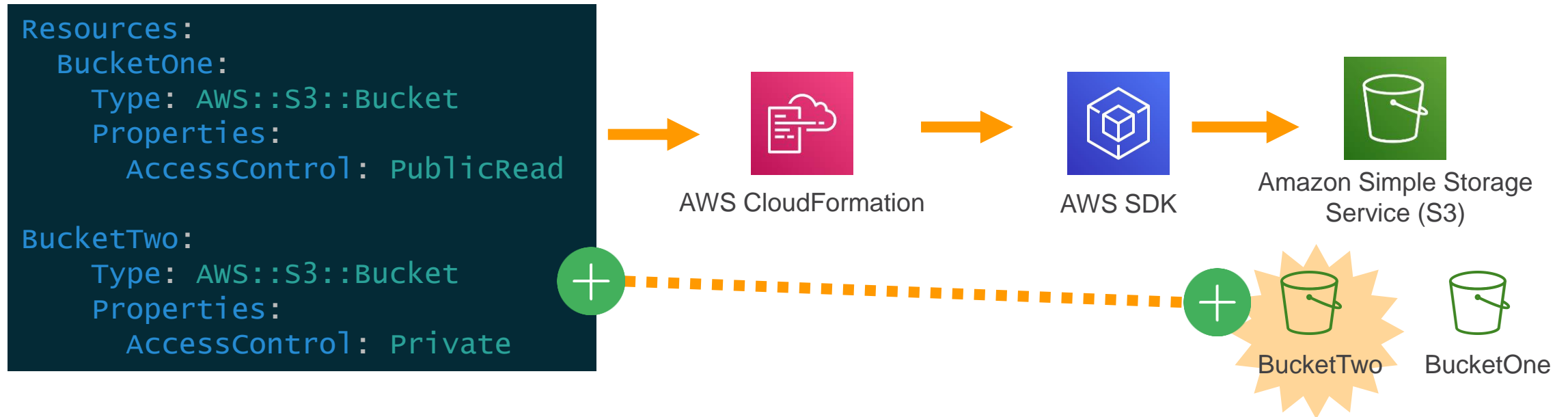
Level 2 – Declarative infrastructure as code

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



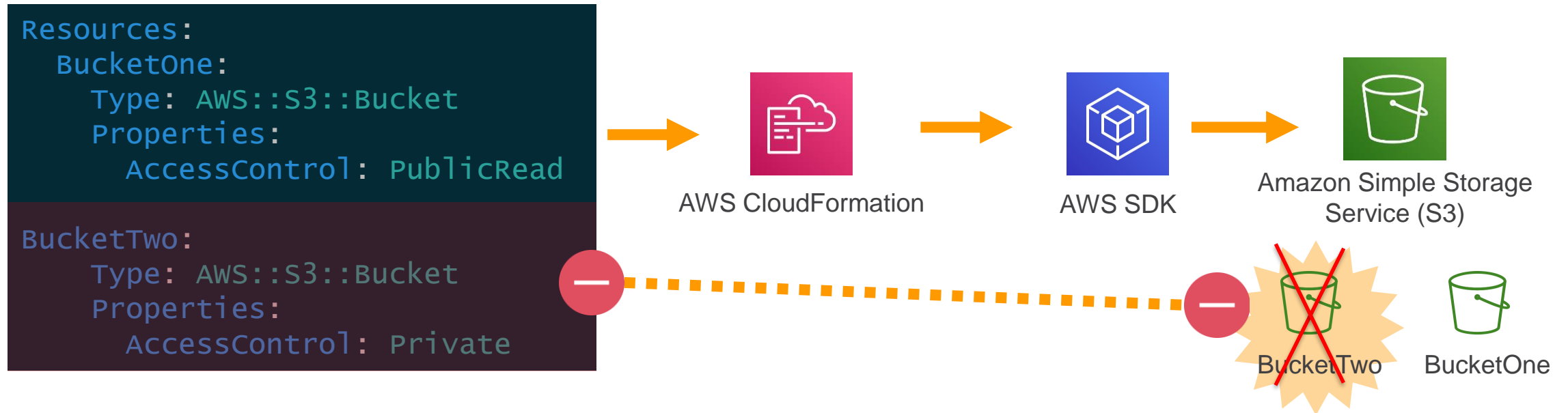
S3 bucket described > Pass to CF > SDK creates bucket in S3

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**

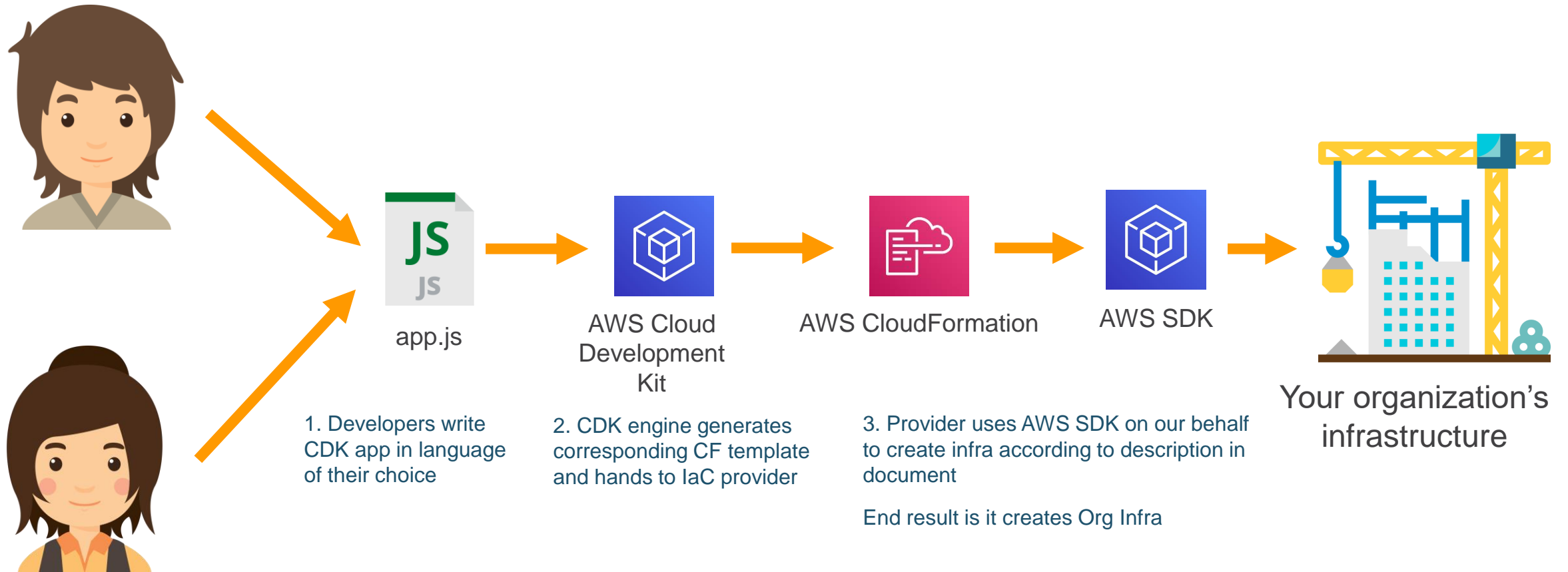


- **1 to 1** relationship between resources in file and resources on account means lots of boilerplate to write. Templates can be very verbose
- Limited ability to run logic as the file formats are generally things like JSON, YAML, or HCL which have only a few built in functions
- Hard to keep things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

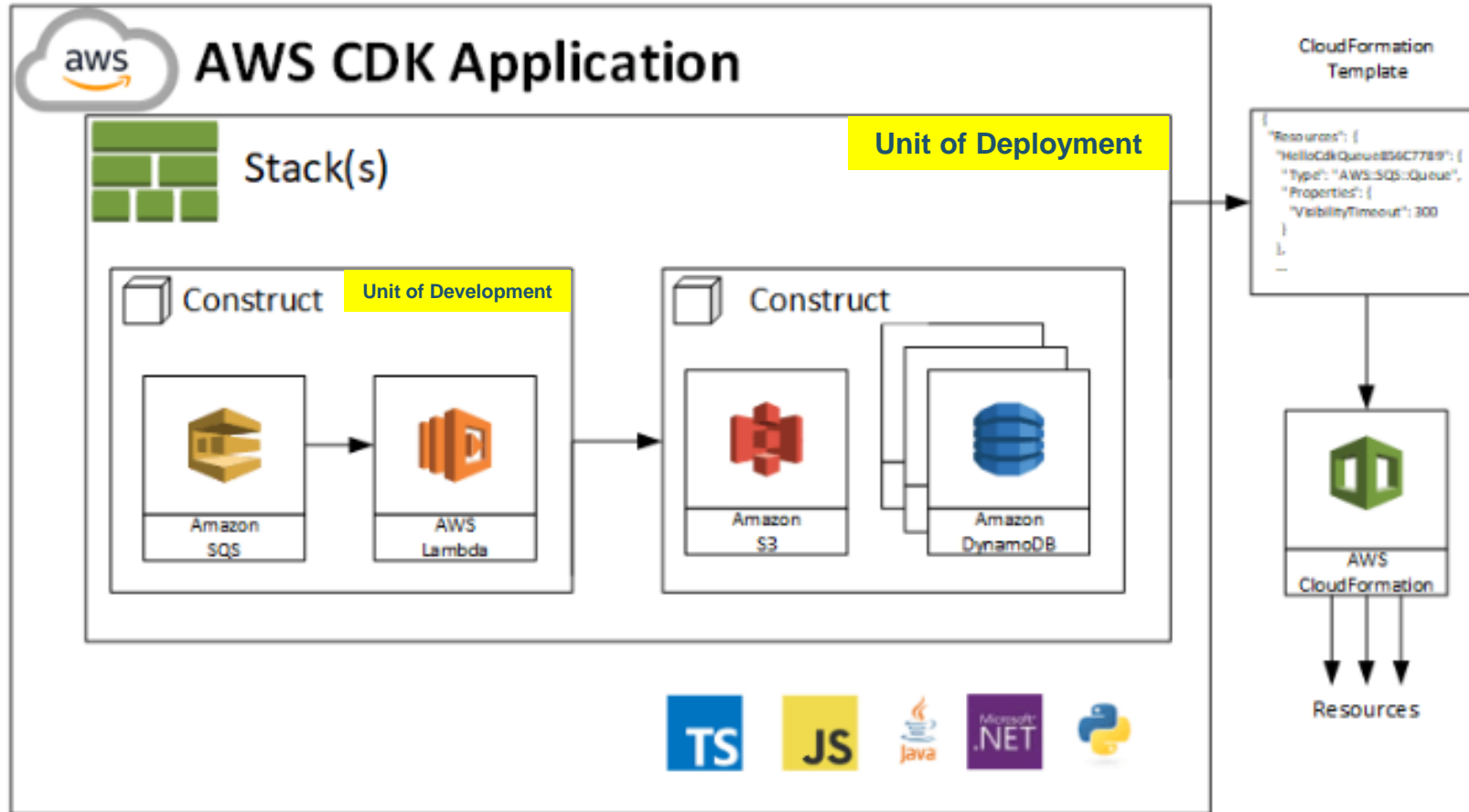


Cons

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

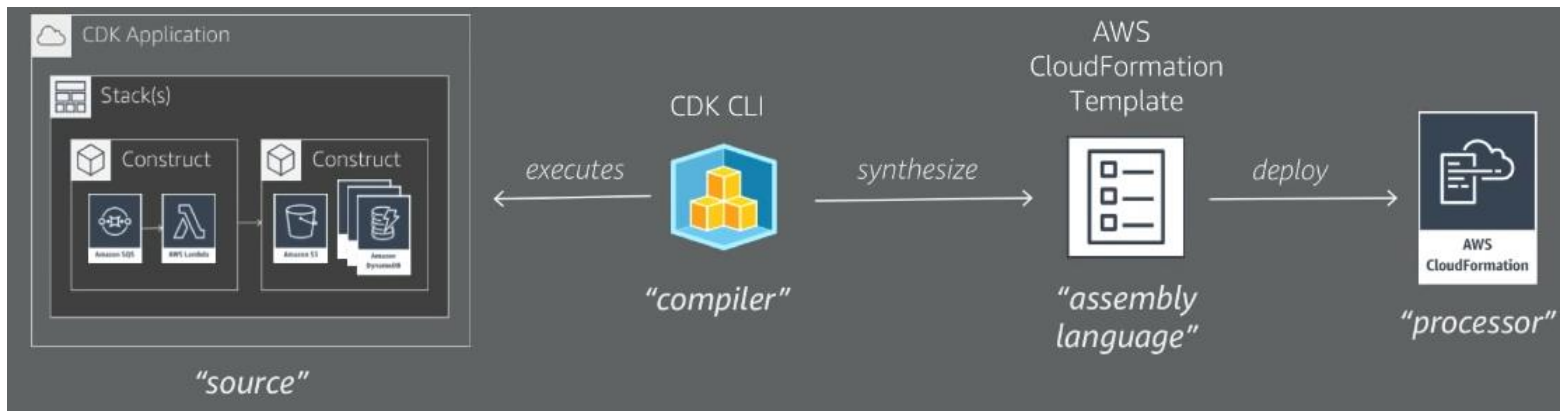
```
class MyService extends cdk.Stack {
  constructor(scope: cdk.App, id: string) {
    super(scope, id);

    // 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



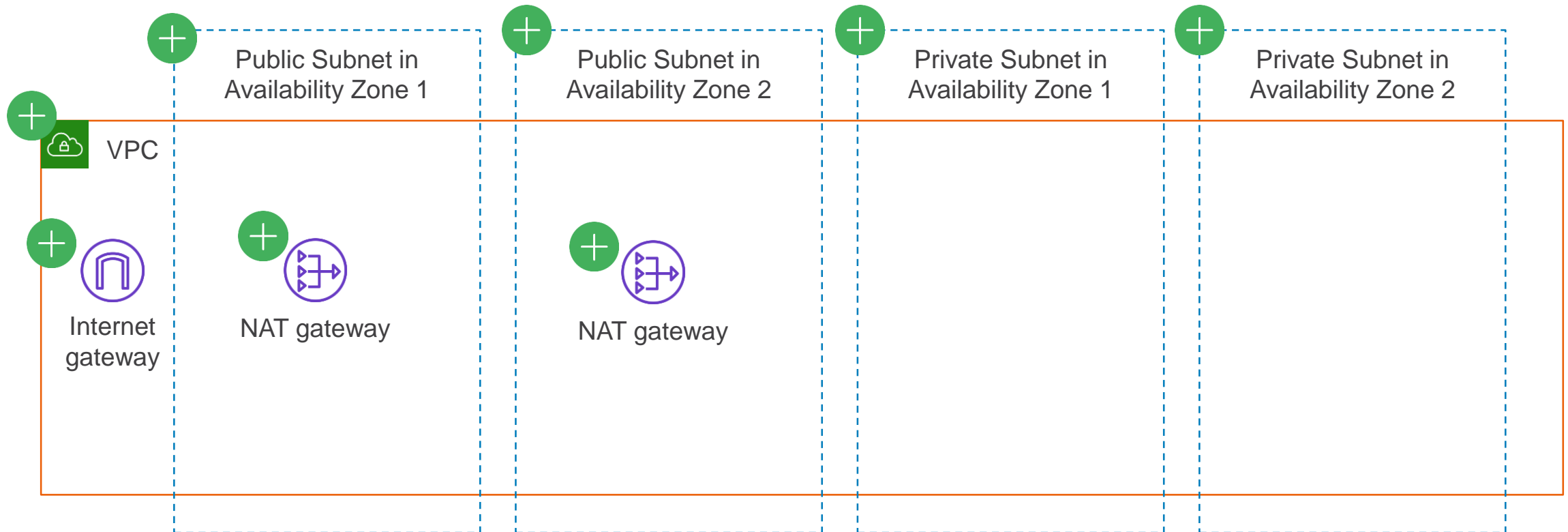
One CDK construct expands to many underlying resources



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

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

cdk deploy



cdk synth (-j)



```

1  Resources:
2  - MyVpc:VPC
3  - Type AWS::EC2::VPC
4  Properties:
5    CidrBlock: 10.0.0.0/16
6    EnableDnsHostnames: true
7    EnableDnsSupport: true
8    InstanceTenancy: default
9    Tags:
10     - Key: Name
11       Value: public-fargate-service/MyVpc
12
13  #Subnets
14  aws-eks-paths: public-fargate-service/MyVpc/Resource
15  Type AWS::EC2::Subnet
16  Properties:
17    CidrBlock: 10.0.0.0/18
18    VpcId:
19      Ref: MyVpc/PFCASf
20    AvailabilityZone:
21      Fn::Select:
22        - Fn::GetAZs: ""
23        - MyPublicSubnet1: true
24    MapPublicIpOnLaunch: true
25    Tags:
26     - Key: Name
27       Value: public-fargate-service/MyVpc/PublicSubnet1
28     - Key: aws-eks-subnet-name
29       Value: Public
30     - Key: aws-eks-subnet-type
31       Value: Public
32
33  #RouteTable
34  aws-eks-paths: public-fargate-service/MyVpc/PublicSubnet1/Subnet
35  MyVpcPublicSubnet1RouteTableASoARFId:
36    Type AWS::EC2::RouteTable
37    Properties:
38      VpcId:
39        Ref: MyVpc/PFCASf
40      Tags:
41       - Key: Name
42         Value: public-fargate-service/MyVpc/PublicSubnet1
43
44  #RouteTableAssociation
45  aws-eks-paths: public-fargate-service/MyVpc/PublicSubnet1/Subnet/RouteTable
46  MyVpcPublicSubnet1RouteTableAssociationASoARFId:
47    Type AWS::EC2::RouteTableAssociation
48    Properties:
49      RouteTableId:
50        Ref: MyVpcPublicSubnet1RouteTableASoARFId
51      SubnetId:
52        Ref: MyVpcPublicSubnet1SubnetEC2Resource
53
54  #Subnet
55  aws-eks-paths: public-fargate-service/MyVpc/PublicSubnet1/Subnet/RouteTableAssociation
56  Type AWS::EC2::Subnet
57  Properties:
58    CidrBlock: 10.0.0.0/18
59    VpcId:
60      Ref: MyVpc/PFCASf
61    AvailabilityZone:
62      Fn::Select:
63        - Fn::GetAZs: ""
64        - MyVpcVPCDefaultAZ: true
65
66  #DefaultRouteTable
67  aws-eks-paths: public-fargate-service/MyVpc/PublicSubnet1/DefaultRouteTable
68  Type AWS::EC2::RouteTable
69  Properties:
70    CidrBlock: 10.0.0.0/16
71    VpcId:
72      Ref: MyVpcPublicSubnet1RouteTableASoARFId
73    Tags:
74     - Key: Name
75       Value: public-fargate-service/MyVpc/PublicSubnet1/DefaultRouteTable
76     - Key: aws-eks-subnet-name
77       Value: Public
78     - Key: aws-eks-subnet-type
79       Value: Public
80
81  #Subnet
82  aws-eks-paths: public-fargate-service/MyVpc/PublicSubnet1/DefaultRouteTable
83  Type AWS::EC2::Subnet
84  Properties:
85    CidrBlock: 10.0.0.0/18
86    VpcId:
87      Ref: MyVpc/PFCASf
88    AvailabilityZone:
89      Fn::Select:
90        - Fn::GetAZs: ""
91        - MyPublicSubnet1: true
92    MapPublicIpOnLaunch: true
93    Tags:
94     - Key: Name
95       Value: public-fargate-service/MyVpc/PublicSubnet1
96     - Key: aws-eks-subnet-name
97       Value: Public
98     - Key: aws-eks-subnet-type
99       Value: Public

```

```

801         = Key MySvcOnSubnetType
802     }
803     Name Public
804 }
805 Metadata
806 {
807     apiVersion: kubernetes.io/v1
808     kind: Service
809     name: public-fargate-service/MySvc/PublicSubnet2/Sumnet
810     namespace: MySvcPublicSubnet2
811     labels: {
812         app: public-fargate-service/MySvc/PublicSubnet2/Sumnet
813     }
814     annotations: {
815         mysvc: {
816             EC2: {
817                 RouteTable
818             }
819         }
820     }
821     selector: {
822         mysvc: {
823             EC2: {
824                 RouteTable
825             }
826         }
827     }
828     ports: {
829         name: MySvcPublicSubnet2
830         port: 80
831         targetPort: 80
832     }
833 }
834 MySvcPublicSubnet2
835 {
836     apiVersion: kubernetes.io/v1
837     kind: Subnet
838     name: MySvcPublicSubnet2
839     namespace: MySvcPublicSubnet2
840     labels: {
841         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
842     }
843     annotations: {
844         mysvc: {
845             EC2: {
846                 RouteTableAssociation
847             }
848         }
849     }
850     selector: {
851         mysvc: {
852             EC2: {
853                 RouteTableAssociation
854             }
855         }
856     }
857     ports: {
858         name: MySvcPublicSubnet2
859         port: 80
860         targetPort: 80
861     }
862 }
863 MySvcPublicSubnet2RouteTable
864 {
865     apiVersion: kubernetes.io/v1
866     kind: RouteTable
867     name: MySvcPublicSubnet2RouteTable
868     namespace: MySvcPublicSubnet2
869     labels: {
870         app: public-fargate-service/MySvc/PublicSubnet2/RouteTable
871     }
872     annotations: {
873         mysvc: {
874             EC2: {
875                 RouteTable
876             }
877         }
878     }
879     selector: {
880         mysvc: {
881             EC2: {
882                 RouteTable
883             }
884         }
885     }
886     ports: {
887         name: MySvcPublicSubnet2RouteTable
888         port: 80
889         targetPort: 80
890     }
891 }
892 MySvcPublicSubnet2RouteTableAssociation
893 {
894     apiVersion: kubernetes.io/v1
895     kind: RouteTableAssociation
896     name: MySvcPublicSubnet2RouteTableAssociation
897     namespace: MySvcPublicSubnet2
898     labels: {
899         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
900     }
901     annotations: {
902         mysvc: {
903             EC2: {
904                 RouteTableAssociation
905             }
906         }
907     }
908     selector: {
909         mysvc: {
910             EC2: {
911                 RouteTableAssociation
912             }
913         }
914     }
915     ports: {
916         name: MySvcPublicSubnet2RouteTableAssociation
917         port: 80
918         targetPort: 80
919     }
920 }
921 MySvcPublicSubnet2RouteTableAssociation
922 {
923     apiVersion: kubernetes.io/v1
924     kind: RouteTableAssociation
925     name: MySvcPublicSubnet2RouteTableAssociation
926     namespace: MySvcPublicSubnet2
927     labels: {
928         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
929     }
930     annotations: {
931         mysvc: {
932             EC2: {
933                 RouteTableAssociation
934             }
935         }
936     }
937     selector: {
938         mysvc: {
939             EC2: {
940                 RouteTableAssociation
941             }
942         }
943     }
944     ports: {
945         name: MySvcPublicSubnet2RouteTableAssociation
946         port: 80
947         targetPort: 80
948     }
949 }
950 MySvcPublicSubnet2RouteTableAssociation
951 {
952     apiVersion: kubernetes.io/v1
953     kind: RouteTableAssociation
954     name: MySvcPublicSubnet2RouteTableAssociation
955     namespace: MySvcPublicSubnet2
956     labels: {
957         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
958     }
959     annotations: {
960         mysvc: {
961             EC2: {
962                 RouteTableAssociation
963             }
964         }
965     }
966     selector: {
967         mysvc: {
968             EC2: {
969                 RouteTableAssociation
970             }
971         }
972     }
973     ports: {
974         name: MySvcPublicSubnet2RouteTableAssociation
975         port: 80
976         targetPort: 80
977     }
978 }
979 MySvcPublicSubnet2RouteTableAssociation
980 {
981     apiVersion: kubernetes.io/v1
982     kind: RouteTableAssociation
983     name: MySvcPublicSubnet2RouteTableAssociation
984     namespace: MySvcPublicSubnet2
985     labels: {
986         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
987     }
988     annotations: {
989         mysvc: {
990             EC2: {
991                 RouteTableAssociation
992             }
993         }
994     }
995     selector: {
996         mysvc: {
997             EC2: {
998                 RouteTableAssociation
999             }
1000         }
1001     }
1002     ports: {
1003         name: MySvcPublicSubnet2RouteTableAssociation
1004         port: 80
1005         targetPort: 80
1006     }
1007 }
1008 MySvcPublicSubnet2RouteTableAssociation
1009 {
1010     apiVersion: kubernetes.io/v1
1011     kind: RouteTableAssociation
1012     name: MySvcPublicSubnet2RouteTableAssociation
1013     namespace: MySvcPublicSubnet2
1014     labels: {
1015         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1016     }
1017     annotations: {
1018         mysvc: {
1019             EC2: {
1020                 RouteTableAssociation
1021             }
1022         }
1023     }
1024     selector: {
1025         mysvc: {
1026             EC2: {
1027                 RouteTableAssociation
1028             }
1029         }
1030     }
1031     ports: {
1032         name: MySvcPublicSubnet2RouteTableAssociation
1033         port: 80
1034         targetPort: 80
1035     }
1036 }
1037 MySvcPublicSubnet2RouteTableAssociation
1038 {
1039     apiVersion: kubernetes.io/v1
1040     kind: RouteTableAssociation
1041     name: MySvcPublicSubnet2RouteTableAssociation
1042     namespace: MySvcPublicSubnet2
1043     labels: {
1044         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1045     }
1046     annotations: {
1047         mysvc: {
1048             EC2: {
1049                 RouteTableAssociation
1050             }
1051         }
1052     }
1053     selector: {
1054         mysvc: {
1055             EC2: {
1056                 RouteTableAssociation
1057             }
1058         }
1059     }
1060     ports: {
1061         name: MySvcPublicSubnet2RouteTableAssociation
1062         port: 80
1063         targetPort: 80
1064     }
1065 }
1066 MySvcPublicSubnet2RouteTableAssociation
1067 {
1068     apiVersion: kubernetes.io/v1
1069     kind: RouteTableAssociation
1070     name: MySvcPublicSubnet2RouteTableAssociation
1071     namespace: MySvcPublicSubnet2
1072     labels: {
1073         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1074     }
1075     annotations: {
1076         mysvc: {
1077             EC2: {
1078                 RouteTableAssociation
1079             }
1080         }
1081     }
1082     selector: {
1083         mysvc: {
1084             EC2: {
1085                 RouteTableAssociation
1086             }
1087         }
1088     }
1089     ports: {
1090         name: MySvcPublicSubnet2RouteTableAssociation
1091         port: 80
1092         targetPort: 80
1093     }
1094 }
1095 MySvcPublicSubnet2RouteTableAssociation
1096 {
1097     apiVersion: kubernetes.io/v1
1098     kind: RouteTableAssociation
1099     name: MySvcPublicSubnet2RouteTableAssociation
1100     namespace: MySvcPublicSubnet2
1101     labels: {
1102         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1103     }
1104     annotations: {
1105         mysvc: {
1106             EC2: {
1107                 RouteTableAssociation
1108             }
1109         }
1110     }
1111     selector: {
1112         mysvc: {
1113             EC2: {
1114                 RouteTableAssociation
1115             }
1116         }
1117     }
1118     ports: {
1119         name: MySvcPublicSubnet2RouteTableAssociation
1120         port: 80
1121         targetPort: 80
1122     }
1123 }
1124 MySvcPublicSubnet2RouteTableAssociation
1125 {
1126     apiVersion: kubernetes.io/v1
1127     kind: RouteTableAssociation
1128     name: MySvcPublicSubnet2RouteTableAssociation
1129     namespace: MySvcPublicSubnet2
1130     labels: {
1131         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1132     }
1133     annotations: {
1134         mysvc: {
1135             EC2: {
1136                 RouteTableAssociation
1137             }
1138         }
1139     }
1140     selector: {
1141         mysvc: {
1142             EC2: {
1143                 RouteTableAssociation
1144             }
1145         }
1146     }
1147     ports: {
1148         name: MySvcPublicSubnet2RouteTableAssociation
1149         port: 80
1150         targetPort: 80
1151     }
1152 }
1153 MySvcPublicSubnet2RouteTableAssociation
1154 {
1155     apiVersion: kubernetes.io/v1
1156     kind: RouteTableAssociation
1157     name: MySvcPublicSubnet2RouteTableAssociation
1158     namespace: MySvcPublicSubnet2
1159     labels: {
1160         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1161     }
1162     annotations: {
1163         mysvc: {
1164             EC2: {
1165                 RouteTableAssociation
1166             }
1167         }
1168     }
1169     selector: {
1170         mysvc: {
1171             EC2: {
1172                 RouteTableAssociation
1173             }
1174         }
1175     }
1176     ports: {
1177         name: MySvcPublicSubnet2RouteTableAssociation
1178         port: 80
1179         targetPort: 80
1180     }
1181 }
1182 MySvcPublicSubnet2RouteTableAssociation
1183 {
1184     apiVersion: kubernetes.io/v1
1185     kind: RouteTableAssociation
1186     name: MySvcPublicSubnet2RouteTableAssociation
1187     namespace: MySvcPublicSubnet2
1188     labels: {
1189         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1190     }
1191     annotations: {
1192         mysvc: {
1193             EC2: {
1194                 RouteTableAssociation
1195             }
1196         }
1197     }
1198     selector: {
1199         mysvc: {
1200             EC2: {
1201                 RouteTableAssociation
1202             }
1203         }
1204     }
1205     ports: {
1206         name: MySvcPublicSubnet2RouteTableAssociation
1207         port: 80
1208         targetPort: 80
1209     }
1210 }
1211 MySvcPublicSubnet2RouteTableAssociation
1212 {
1213     apiVersion: kubernetes.io/v1
1214     kind: RouteTableAssociation
1215     name: MySvcPublicSubnet2RouteTableAssociation
1216     namespace: MySvcPublicSubnet2
1217     labels: {
1218         app: public-fargate-service/MySvc/PublicSubnet2/RouteTableAssociation
1219     }
1220     annotations: {
1221         mysvc: {
1222            
```

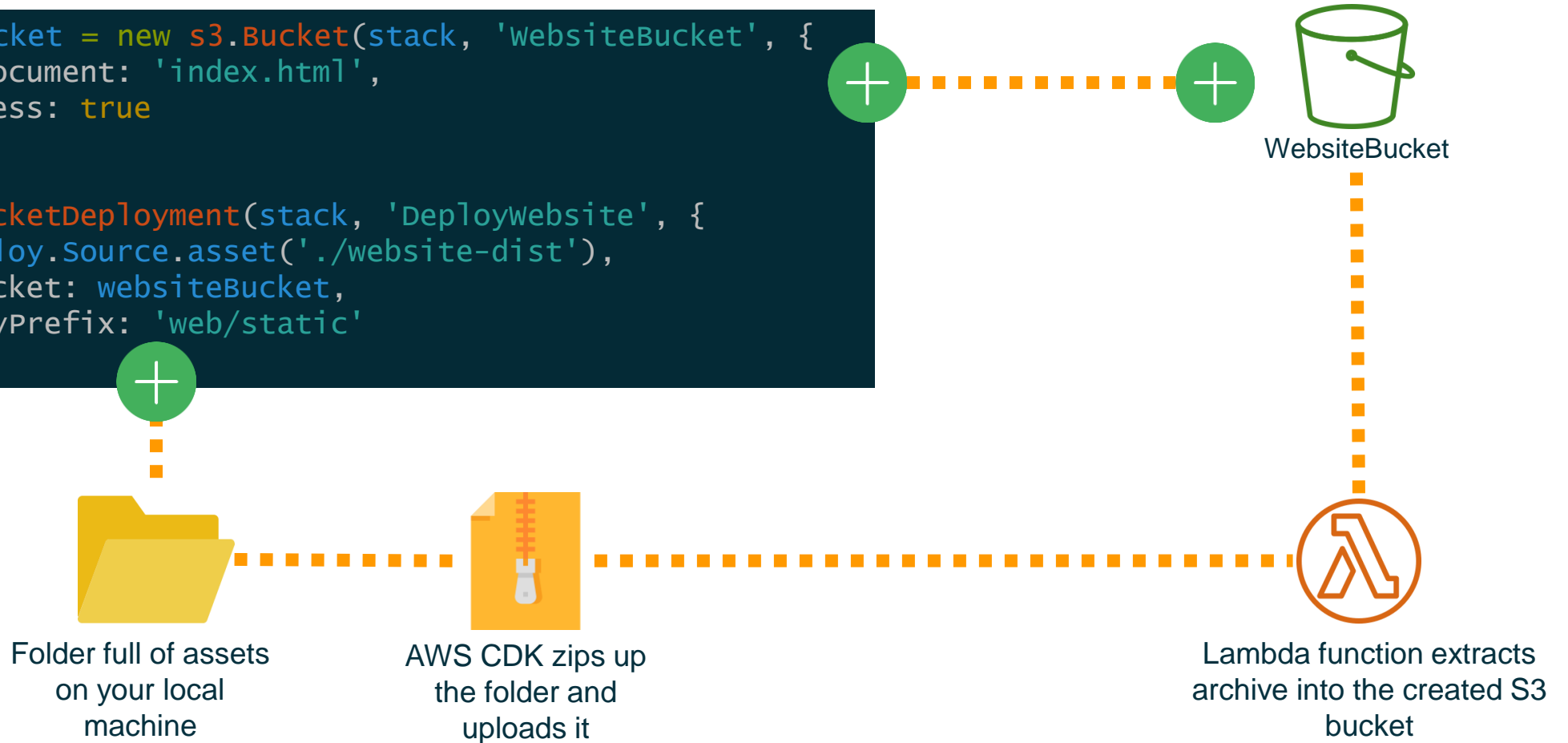
```

280 DestinationIpPrefix: 0.0.0/0
281 NatGatewayId:
282   Ref: MyVpcPublicSubnet3NATGatewayA03488C1
283 Metadata:
284   aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet1/defaultRoute
285 MyVpcPrivateSubnet2Subnet084AC9B3:
286   Type: AWS::EC2::Subnet
287   Properties:
288     CidrBlock: 10.0.132.0/18
289     VpcId:
290       Ref: MyVpcP9F8CA6F
291     AvailabilityZone:
292       Fn::Select:
293         - 1
294         - Fn::GetAZs: ""
295     MapPublicIpOnLaunch: false
296     Tags:
297       - Key: Name
298         Value: public-fargate-service/MyVpc/PrivateSubnet2
299       - Key: aws:cdk:subnet-name
300         Value: Private
301       - Key: aws-cdk:subnet-type
302         Value: Private
303 Metadata:
304   aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/Subnet
305 MyVpcPrivateSubnet2RouteTableECDECEEC:
306   Type: AWS::EC2::RouteTable
307   Properties:
308     VpcId:
309       Ref: MyVpcP9F8CA6F
310     Tags:
311       - Key: Name
312         Value: public-fargate-service/MyVpc/PrivateSubnet2
313 Metadata:
314   aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/RouteTable
315 MyVpcPrivateSubnet2RouteTableAssociation08A6180A:
316   Type: AWS::EC2::SubnetRouteTableAssociation
317   Properties:
318     RouteTableId:
319       Ref: MyVpcPrivateSubnet2RouteTableECDECEE
320     SubnetId:
321       Ref: MyVpcPrivateSubnet2Subnet084AC9B3
322 Metadata:
323   aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/RouteTableAssociation
324 MyVpcPrivateSubnet2DefaultRouteEC962941:
325   Type: AWS::EC2::Route
326   Properties:
327     RouteTableId:
328       Ref: MyVpcPrivateSubnet2RouteTableECDECEE
329     DestinationCIDRBlock: 0.0.0/0
330     NatGatewayId:
331       Ref: MyVpcPublicSubnet2NATGateway918BECC9
332 Metadata:
333   aws:cdk:path: public-fargate-service/MyVpc/PrivateSubnet2/defaultRoute
334 MyVpcIGW5C4AF63:
335   Type: AWS::EC2::InternetGateway
336   Properties:
337     Tags:
338       - Key: Name
339         Value: public-fargate-service/MyVpc
340 Metadata:
341   aws:cdk:path: public-fargate-service/MyVpc/IGW
342 MyVpcVPCGW48BACE00:
343   Type: AWS::EC2::VPCGatewayAttachment
344   Properties:
345     VpcId:
346       Ref: MyVpcP9F8CA6F
347     InternetGatewayId:
348       Ref: MyVpcIGW5C4AF63
349 Metadata:
350   aws:cdk:path: public-fargate-service/MyVpc/VPCGW

```

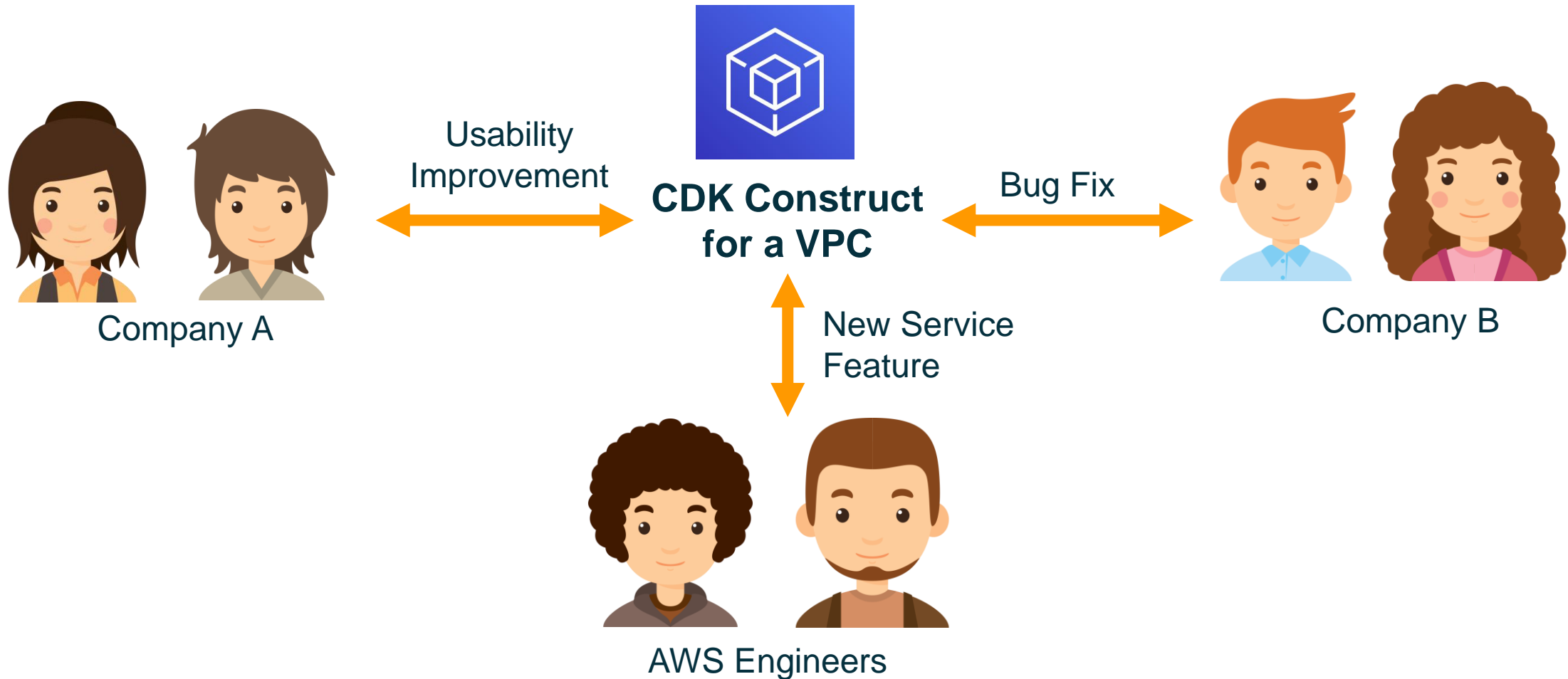
CDK helps with your local workflow too

```
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



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

Two levels of container abstraction in CDK

@aws-cdk/aws-ecs

1.7.0 • Public • Published 4 days ago

Readme

23 Dependencies

Amazon ECS Construct Library

STABILITY STABLE

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

@aws-cdk/aws-ecs-patterns

1.15.0 • Public • Published 2 hours ago

Readme

13 Dependencies

CDK Construct library for higher-level ECS Constructs

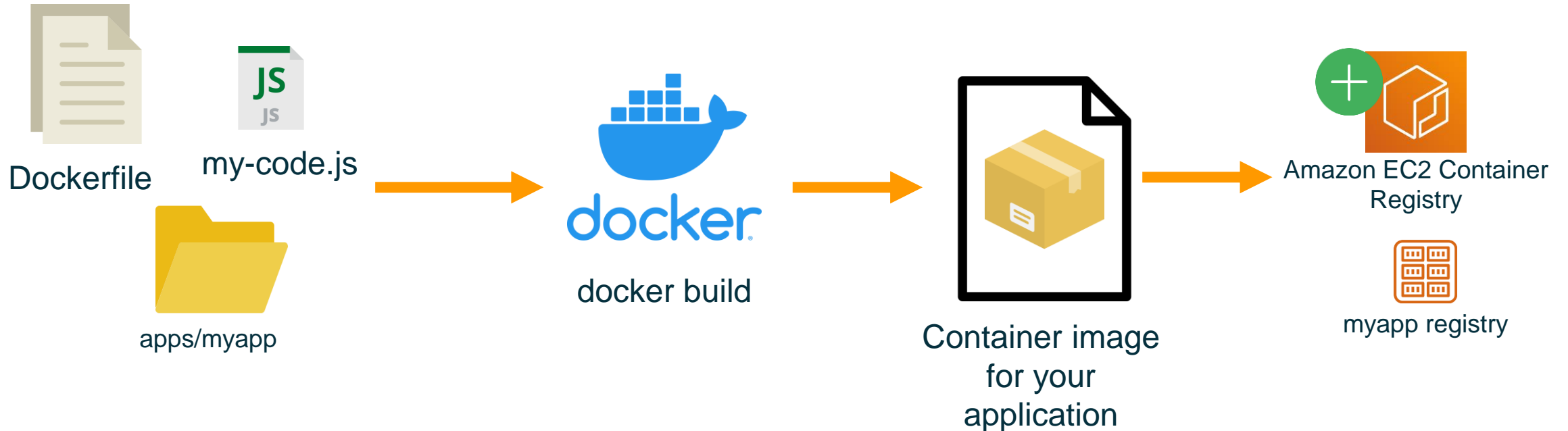
STABILITY STABLE

- 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.
- Stable release

@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
});
```

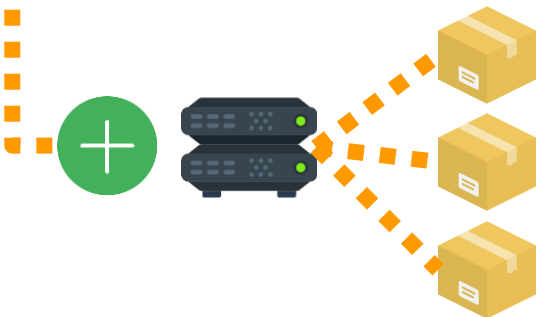
@aws-cdk/aws-ecs-patterns: Load balanced service

```
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



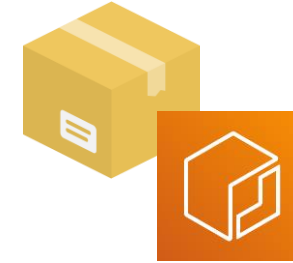
With a few lines we are automatically **building** a Docker container locally, **pushing** it up to the cloud in an Amazon Elastic Container Registry, then **launching** running three copies of it in AWS Fargate, **behind** a load balancer that distributes traffic across all three.

Let's dive a little deeper now

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

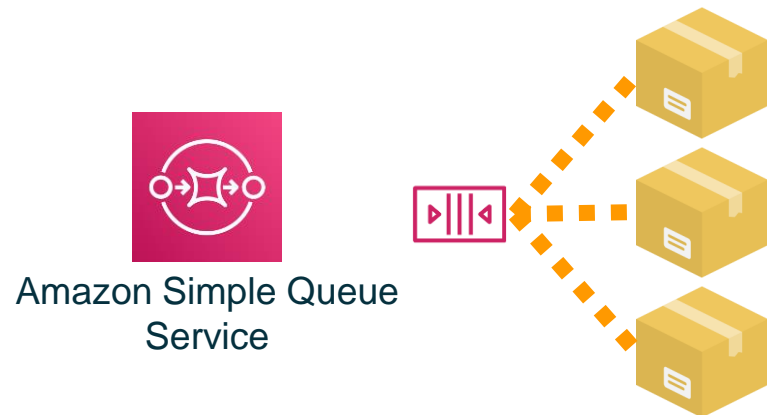
Next steps

- 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>

@aws-cdk/aws-ecs-patterns: Queue consumer

```
const queue = new sqs.Queue(stack);

const consumer = new ecs_patterns.QueueProcessingFargateService(stack, "consumer", {
  cluster,
  queue,
  desiredTaskCount: 3,
  image: ecs.ContainerImage.fromAsset("apps/consumer")
});
```



Create an SQS queue, plus a service which autoscales according to how many items are waiting in the queue. If the queue backs up more containers are launched to grab items off the queue.

@aws-cdk/aws-ecs-patterns: Time scheduled container

```
const ecsScheduledTask = new ScheduledFargateTask(stack, 'ScheduledTask', {
  cluster,
  image: ecs.ContainerImage.fromRegistry("apps/my-cron-job"),
  scheduleExpression: 'rate(1 day)',
  environment: [{ name: 'TRIGGER', value: 'CloudWatch Events' }],
  memoryLimitMiB: 256
});
```



Amazon CloudWatch



Every day at
5:00



Execute the container based on a scheduled time or rate.
High availability, low cost distributed cron jobs!