

Deploying Containerized 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?
 - *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?

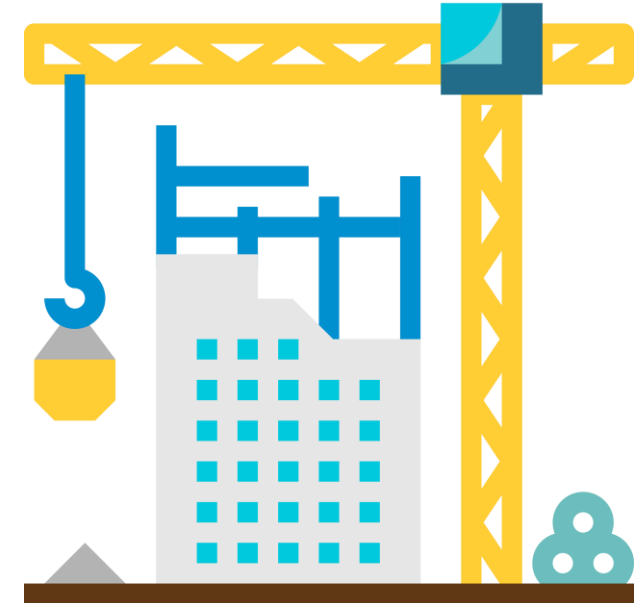
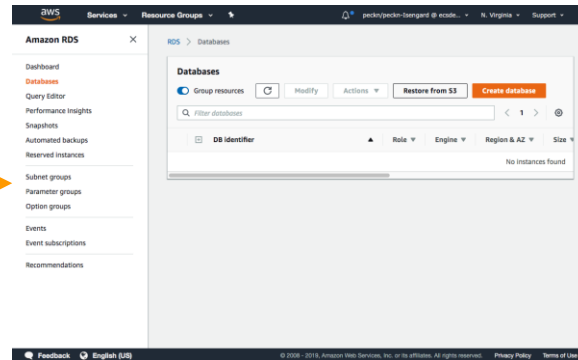
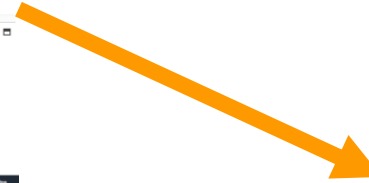
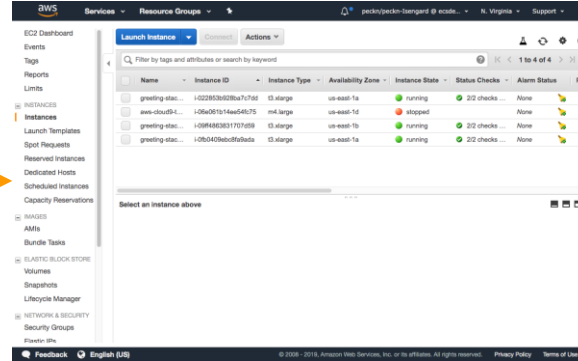
- High level reusable patterns for containers
 - *aws-ecs-patterns (Beginner, High-level), aws-ecs (Advanced, Low-level) constructs*
- Underlying core constructs for more advanced configurations

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



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

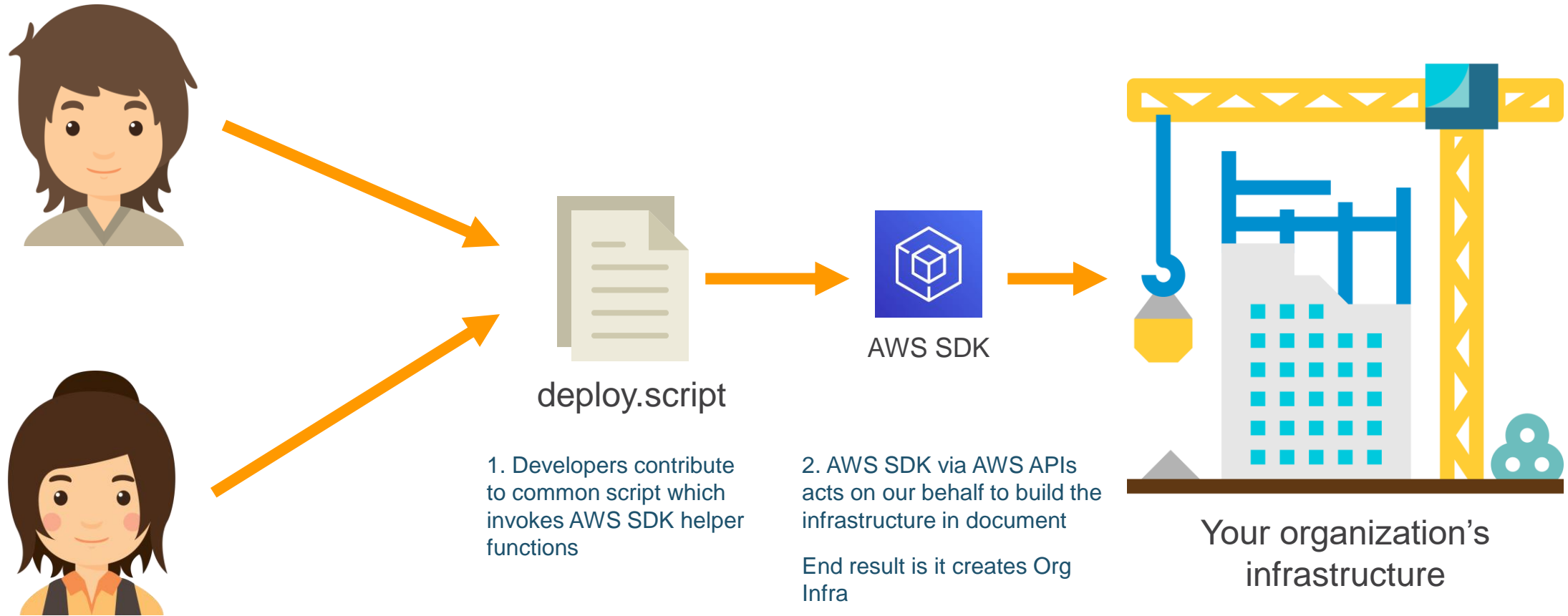


- Clicking things and entering values in the console by hand is **slow**. **Bottleneck** for long-term development
- It's hard to reliably **reproduce** your results when you are doing things manually, by hand.
- People make **mistakes** in data entry and clicking options.
Can't standardize reliability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

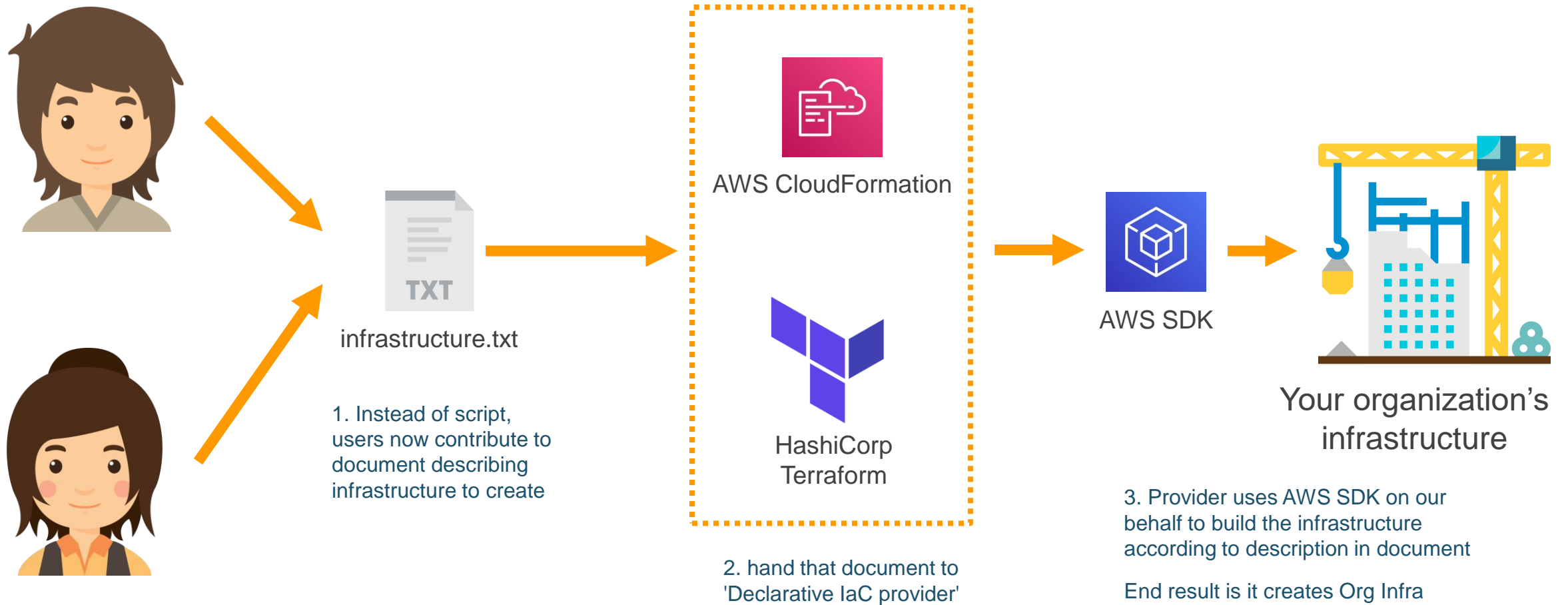
- If the code is written well it is repeatable and reusable
- Multiple people can work on the script collaboratively, fixing bugs, see all the settings in one place



Cons

- 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

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

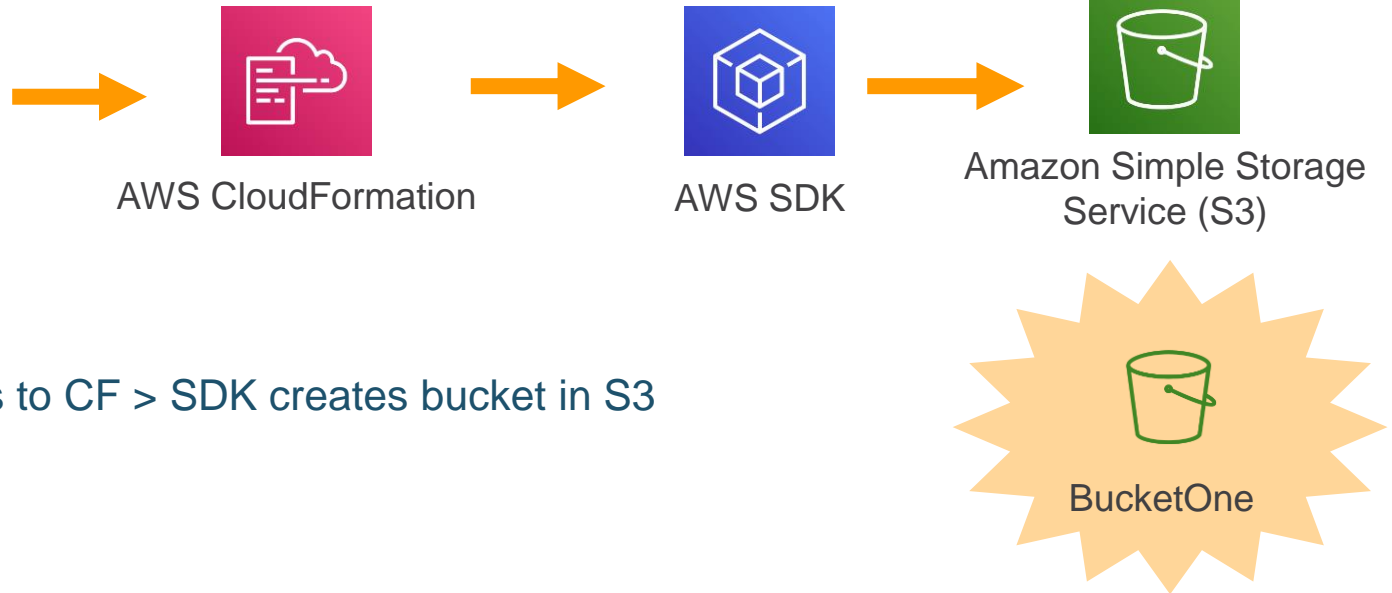
```
Resources:
  # VPC in which containers will be networked.
  # It has two public subnets
  # We distribute the subnets across the first two available subnets
  # for the region, for high availability.
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: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

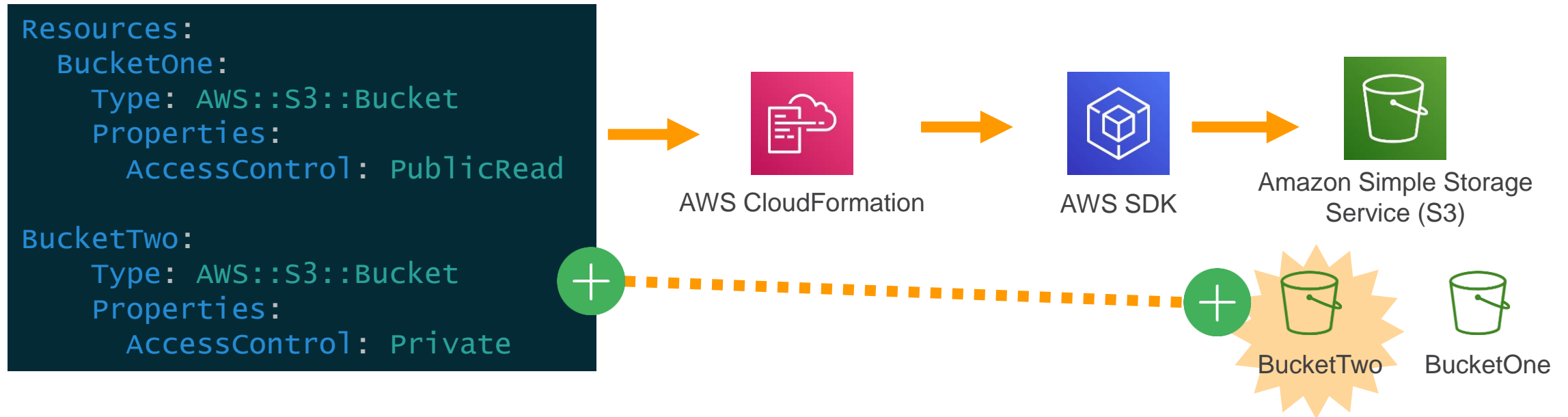
Level 2 – Declarative infrastructure as code

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



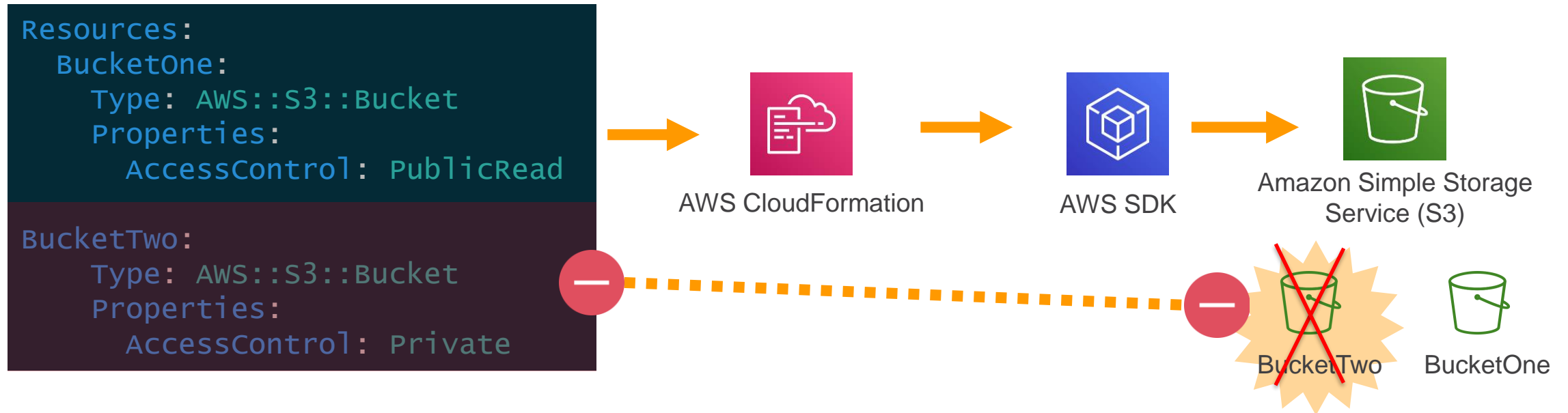
S3 bucket described > Pass to CF > SDK creates bucket in S3

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**

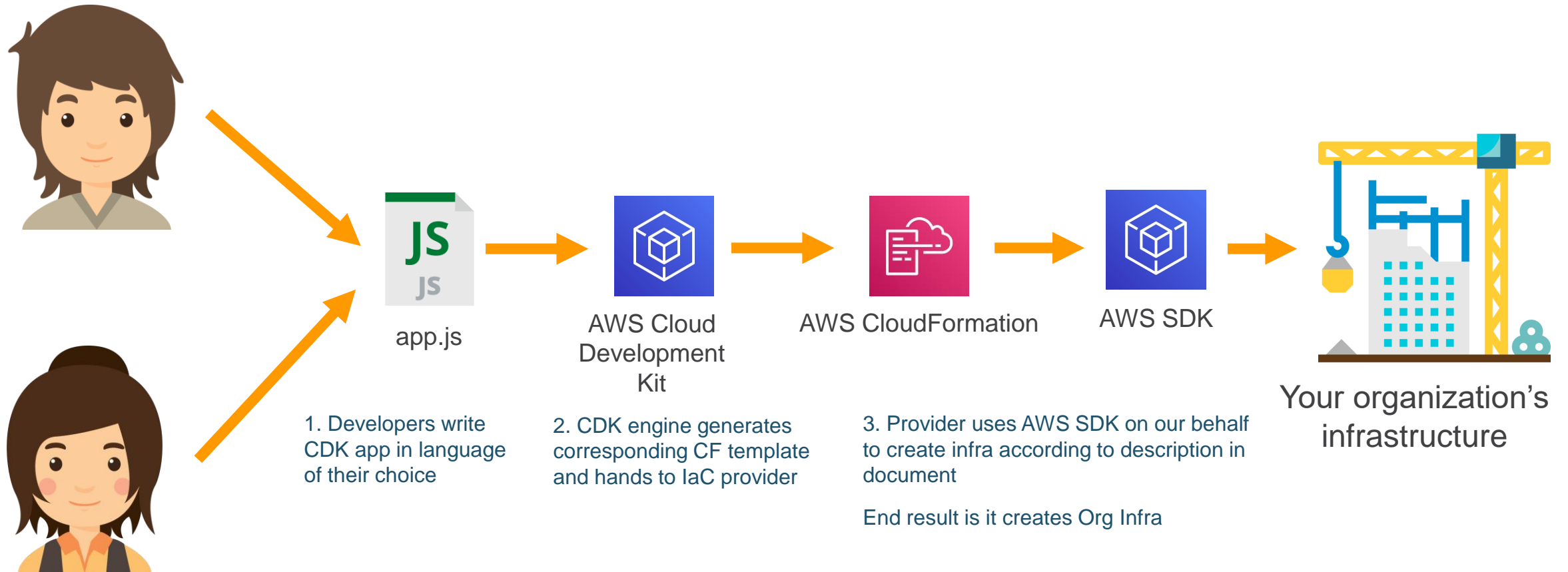


- **1 to 1** relationship between resources in file and resources on account means lots of boilerplate to write. Templates can be very verbose
- Limited ability to run logic as the file formats are generally things like JSON, YAML, or HCL which have only a few built in functions
- Hard to keep things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

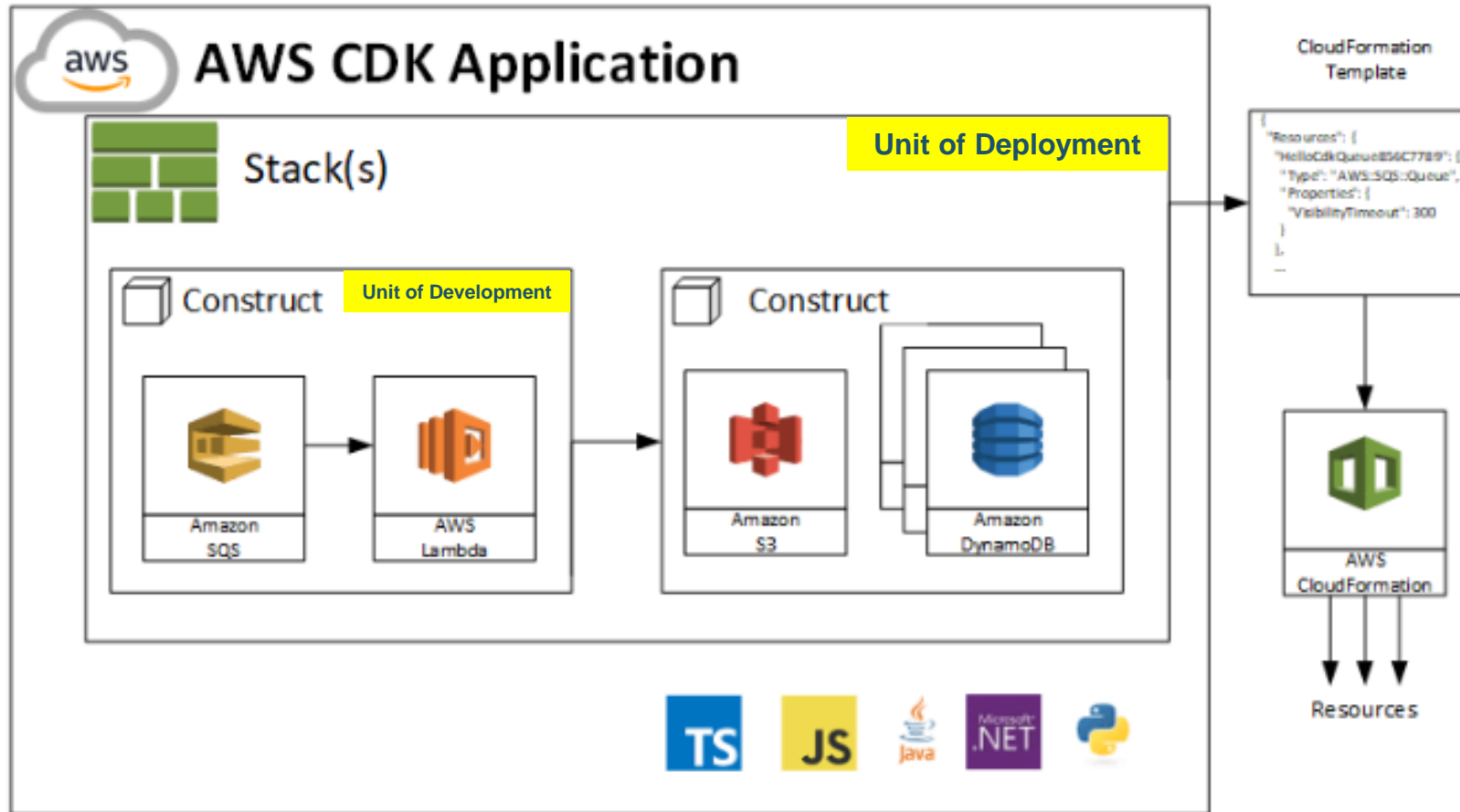


Cons

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

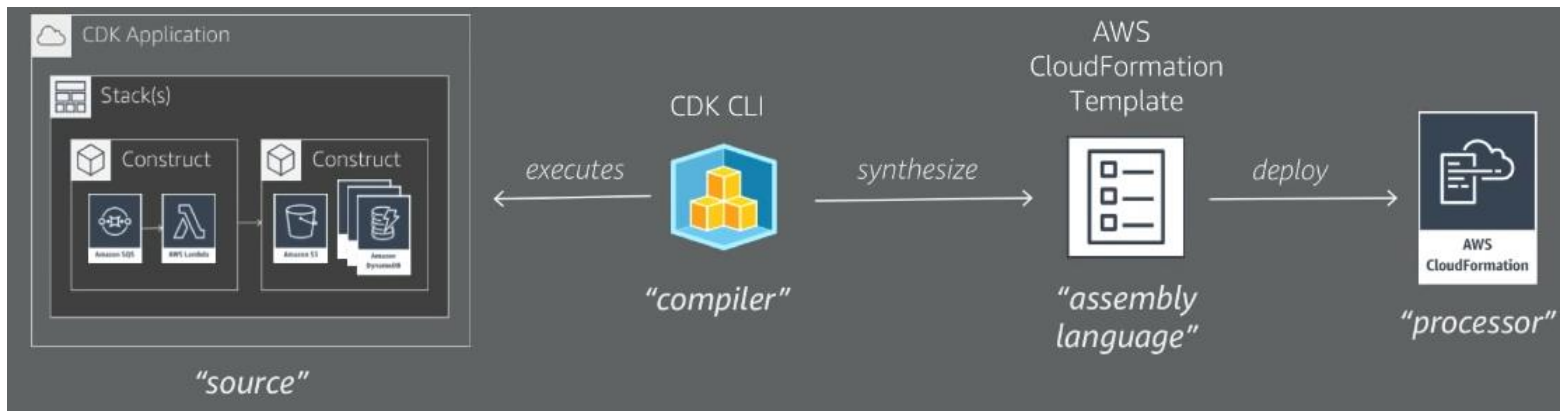
```
class MyService extends cdk.Stack {
  constructor(scope: cdk.App, id: string) {
    super(scope, id);

    // 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



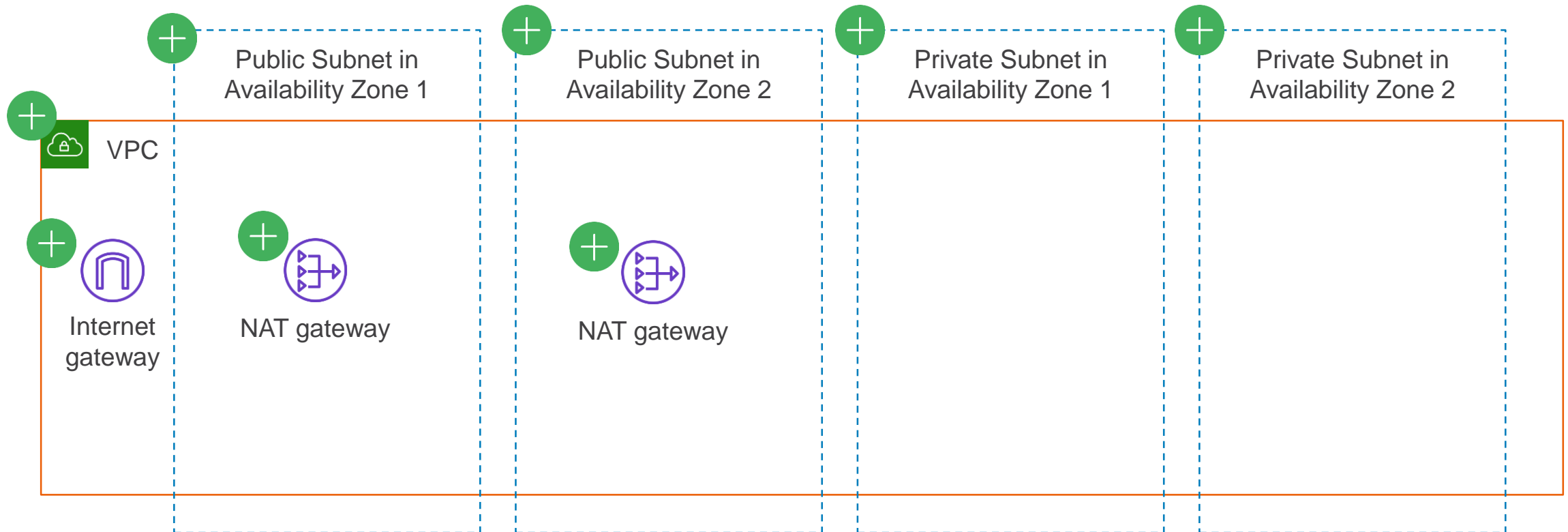
One CDK construct expands to many underlying resources



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

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

cdk deploy





cdk synth (-j)



```

1  Resources:
2  - MyApp::EC2::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/MyApp
12
13  #Subnets
14  aws-eks-paths: public-fargate-service/MyApp/Resource
15  Type AWS::EC2::Subnet
16  Properties:
17    - CidrBlock: 10.0.0.0/16
18    - VpcId:
19      Ref: MyApp::PFCASf
20    - AvailabilityZone:
21      Fn::Select:
22        - 0
23        - Fn::GetAZs: ""
24    - MapPublicIpOnLaunch: true
25  Tags:
26    - Key: Name
27      Value: public-fargate-service/MyApp/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/MyApp/PublicSubnet1/Subnet
35  MyApp::PublicSubnet1RouteTableASoARFId:
36  Type AWS::EC2::RouteTable
37  Properties:
38    - VpcId:
39      Ref: MyApp::PFCASf
40    - Tags:
41      - Key: Name
42        Value: public-fargate-service/MyApp/PublicSubnet1
43
44  #RouteTableAssociation
45  aws-eks-paths: public-fargate-service/MyApp/PublicSubnet1/Subnet/RouteTable
46  MyApp::PublicSubnet1RouteTableAssociationEC2EC2Id:
47  Type AWS::EC2::RouteTableAssociation
48  Properties:
49    - RouteTableId:
50      Ref: MyApp::PublicSubnet1RouteTableASoARFId
51    - SubnetId:
52      Ref: MyApp::PublicSubnet1SubnetEC2EC2Id
53
54  #Subnet
55  aws-eks-paths: public-fargate-service/MyApp/PublicSubnet1/Subnet/RouteTableAssociation
56  Type AWS::EC2::Subnet
57  Properties:
58    - CidrBlock: 10.0.0.0/16
59    - VpcId:
60      Ref: MyApp::PFCASf
61    - AvailabilityZone:
62      Fn::Select:
63        - 0
64        - Fn::GetAZs: ""
65    - MapPublicIpOnLaunch: true
66  Tags:
67    - Key: Name
68      Value: public-fargate-service/MyApp/PublicSubnet1/DefaultRouteTable
69    - Key: aws-eks-subnet-name
70      Value: Public
71    - Key: aws-eks-subnet-type
72      Value: Public
73
74  #Subnet
75  aws-eks-paths: public-fargate-service/MyApp/PublicSubnet1/ZIP
76  MyApp::PublicSubnet1SubnetMyAppPFCASfId:
77  Type AWS::EC2::Subnet
78  Properties:
79    - CidrBlock: 10.0.0.0/16
80    - VpcId:
81      Ref: MyApp::PFCASf
82    - AvailabilityZone:
83      Fn::Select:
84        - 0
85        - Fn::GetAZs: ""
86    - MapPublicIpOnLaunch: true
87  Tags:
88    - Key: Name
89      Value: public-fargate-service/MyApp/PublicSubnet1
90    - Key: aws-eks-subnet-name
91      Value: Public

```

```

801         = Key MySvcOnSubnetType
802         Name Public
803     Metadata
804         mysvc:public-fargate-service/MySvc/PublicSubnet2/Subnet
805     mysvc:public-fargate-service/MySvc/PublicSubnet2/Subnet
806     Type AWS::EC2::RouteTable
807     Properties
808         VpcId      = Ref::VpcPublicSubnet1
809         RouteTableId
810         Tags
811             = Key Name
812             Value public-fargate-service/MySvc/PublicSubnet2
813     Metadata
814         mysvc:public-fargate-service/MySvc/PublicSubnet2/RouteTable
815     mysvc:public-fargate-service/MySvc/PublicSubnet2/RouteTable
816     Type AWS::EC2::RouteTable
817     Properties
818         VpcId      = Ref::VpcPublicSubnet1
819         RouteTableId
820         Tags
821             = Key Name
822             Value public-fargate-service/MySvc/PublicSubnet2/RouteTable/Association
823     mysvc:public-fargate-service/MySvc/PublicSubnet2/RouteTable/Association
824     Type AWS::EC2::Route
825     Properties
826         RouteTableId
827         VpcId      = Ref::VpcPublicSubnet1
828         Destination = Ref::VpcPublicSubnet2
829         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
830         GatewayId   = Ref::VpcVpnGateway
831         DependsOn   = [MySvcVpnGateway]
832     Metadata
833         mysvc:public-fargate-service/MySvc/PublicSubnet2/DefaultRoute
834     mysvc:public-fargate-service/MySvc/PublicSubnet2/DefaultRoute
835     Type AWS::EC2::EIP
836     Properties
837         Domain = vpc
838     Metadata
839         mysvc:public-fargate-service/MySvc/PublicSubnet2/EIP
840     mysvc:public-fargate-service/MySvc/PublicSubnet2/EIP
841     Type AWS::EC2::ElasticIP
842     Properties
843         AllocationId
844         VpcId      = Ref::VpcPublicSubnet2
845         SubnetId    = Ref::VpcPublicSubnet2
846         DependsOn   = [MySvcVpnGateway]
847     Metadata
848         mysvc:public-fargate-service/MySvc/PublicSubnet2/DefaultRoute
849     mysvc:public-fargate-service/MySvc/PublicSubnet2/DefaultRoute
850     Type AWS::EC2::RouteTable
851     Properties
852         VpcId      = Ref::VpcPublicSubnet1
853         RouteTableId
854     Metadata
855         mysvc:public-fargate-service/MySvc/PublicSubnet2/PrivateSubnet
856     mysvc:private-subnet/SubnetPrivate1
857     Type AWS::EC2::Subnet
858     Properties
859         CidrBlock = Ref::VpcSubnet1B
860         VpcId     = Ref::VpcPublicSubnet1
861         Tags
862             = Key Name
863             Value public-fargate-service/MySvc/PrivateSubnet1
864     Metadata
865         mysvc:public-fargate-service/MySvc/PublicSubnet2/PrivateSubnet
866     mysvc:public-fargate-service/MySvc/PrivateSubnet1/PrivateSubnet
867     Type AWS::EC2::Subnet
868     Properties
869         VpcId      = Ref::VpcPublicSubnet1
870         RouteTableId
871         Tags
872             = Key Name
873             Value public-fargate-service/MySvc/PrivateSubnet1/RouteTable
874     Metadata
875         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable
876     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable
877     Type AWS::EC2::RouteTable
878     Properties
879         VpcId      = Ref::VpcPublicSubnet1
880         RouteTableId
881         Tags
882             = Key Name
883             Value public-fargate-service/MySvc/PrivateSubnet1
884     Metadata
885         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
886     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
887     Type AWS::EC2::Route
888     Properties
889         RouteTableId
890         VpcId      = Ref::VpcPublicSubnet1
891         Destination = Ref::VpcPublicSubnet2
892         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
893         GatewayId   = Ref::VpcVpnGateway
894         DependsOn   = [MySvcVpnGateway]
895     Metadata
896         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
897     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
898     Type AWS::EC2::Route
899     Properties
900         RouteTableId
901         VpcId      = Ref::VpcPublicSubnet1
902         Destination = Ref::VpcPublicSubnet2
903         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
904         GatewayId   = Ref::VpcVpnGateway
905         DependsOn   = [MySvcVpnGateway]
906     Metadata
907         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
908     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
909     Type AWS::EC2::Route
910     Properties
911         RouteTableId
912         VpcId      = Ref::VpcPublicSubnet1
913         Destination = Ref::VpcPublicSubnet2
914         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
915         GatewayId   = Ref::VpcVpnGateway
916         DependsOn   = [MySvcVpnGateway]
917     Metadata
918         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
919     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
920     Type AWS::EC2::Route
921     Properties
922         RouteTableId
923         VpcId      = Ref::VpcPublicSubnet1
924         Destination = Ref::VpcPublicSubnet2
925         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
926         GatewayId   = Ref::VpcVpnGateway
927         DependsOn   = [MySvcVpnGateway]
928     Metadata
929         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
930     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
931     Type AWS::EC2::Route
932     Properties
933         RouteTableId
934         VpcId      = Ref::VpcPublicSubnet1
935         Destination = Ref::VpcPublicSubnet2
936         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
937         GatewayId   = Ref::VpcVpnGateway
938         DependsOn   = [MySvcVpnGateway]
939     Metadata
940         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
941     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
942     Type AWS::EC2::Route
943     Properties
944         RouteTableId
945         VpcId      = Ref::VpcPublicSubnet1
946         Destination = Ref::VpcPublicSubnet2
947         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
948         GatewayId   = Ref::VpcVpnGateway
949         DependsOn   = [MySvcVpnGateway]
950     Metadata
951         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
952     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
953     Type AWS::EC2::Route
954     Properties
955         RouteTableId
956         VpcId      = Ref::VpcPublicSubnet1
957         Destination = Ref::VpcPublicSubnet2
958         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
959         GatewayId   = Ref::VpcVpnGateway
960         DependsOn   = [MySvcVpnGateway]
961     Metadata
962         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
963     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
964     Type AWS::EC2::Route
965     Properties
966         RouteTableId
967         VpcId      = Ref::VpcPublicSubnet1
968         Destination = Ref::VpcPublicSubnet2
969         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
970         GatewayId   = Ref::VpcVpnGateway
971         DependsOn   = [MySvcVpnGateway]
972     Metadata
973         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
974     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
975     Type AWS::EC2::Route
976     Properties
977         RouteTableId
978         VpcId      = Ref::VpcPublicSubnet1
979         Destination = Ref::VpcPublicSubnet2
980         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
981         GatewayId   = Ref::VpcVpnGateway
982         DependsOn   = [MySvcVpnGateway]
983     Metadata
984         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
985     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
986     Type AWS::EC2::Route
987     Properties
988         RouteTableId
989         VpcId      = Ref::VpcPublicSubnet1
990         Destination = Ref::VpcPublicSubnet2
991         CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
992         GatewayId   = Ref::VpcVpnGateway
993         DependsOn   = [MySvcVpnGateway]
994     Metadata
995         mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
996     mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
997     Type AWS::EC2::Route
998     Properties
999         RouteTableId
1000        VpcId      = Ref::VpcPublicSubnet1
1001        Destination = Ref::VpcPublicSubnet2
1002        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1003        GatewayId   = Ref::VpcVpnGateway
1004        DependsOn   = [MySvcVpnGateway]
1005    Metadata
1006        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1007    mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1008    Type AWS::EC2::Route
1009    Properties
1010        RouteTableId
1011        VpcId      = Ref::VpcPublicSubnet1
1012        Destination = Ref::VpcPublicSubnet2
1013        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1014        GatewayId   = Ref::VpcVpnGateway
1015        DependsOn   = [MySvcVpnGateway]
1016    Metadata
1017        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1018    mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1019    Type AWS::EC2::Route
1020    Properties
1021        RouteTableId
1022        VpcId      = Ref::VpcPublicSubnet1
1023        Destination = Ref::VpcPublicSubnet2
1024        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1025        GatewayId   = Ref::VpcVpnGateway
1026        DependsOn   = [MySvcVpnGateway]
1027    Metadata
1028        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1029    mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1030    Type AWS::EC2::Route
1031    Properties
1032        RouteTableId
1033        VpcId      = Ref::VpcPublicSubnet1
1034        Destination = Ref::VpcPublicSubnet2
1035        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1036        GatewayId   = Ref::VpcVpnGateway
1037        DependsOn   = [MySvcVpnGateway]
1038    Metadata
1039        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1040    mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1041    Type AWS::EC2::Route
1042    Properties
1043        RouteTableId
1044        VpcId      = Ref::VpcPublicSubnet1
1045        Destination = Ref::VpcPublicSubnet2
1046        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1047        GatewayId   = Ref::VpcVpnGateway
1048        DependsOn   = [MySvcVpnGateway]
1049    Metadata
1050        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1051    mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1052    Type AWS::EC2::Route
1053    Properties
1054        RouteTableId
1055        VpcId      = Ref::VpcPublicSubnet1
1056        Destination = Ref::VpcPublicSubnet2
1057        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1058        GatewayId   = Ref::VpcVpnGateway
1059        DependsOn   = [MySvcVpnGateway]
1060    Metadata
1061        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1062    mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1063    Type AWS::EC2::Route
1064    Properties
1065        RouteTableId
1066        VpcId      = Ref::VpcPublicSubnet1
1067        Destination = Ref::VpcPublicSubnet2
1068        CidrBlock   = Ref::VpcPublicSubnet2CidrBlock
1069        GatewayId   = Ref::VpcVpnGateway
1070        DependsOn   = [MySvcVpnGateway]
1071    Metadata
1072        mysvc:public-fargate-service/MySvc/PrivateSubnet1/RouteTable/Association
1073    mysvc:public-fargate-service/MySvc/PrivateSub
```

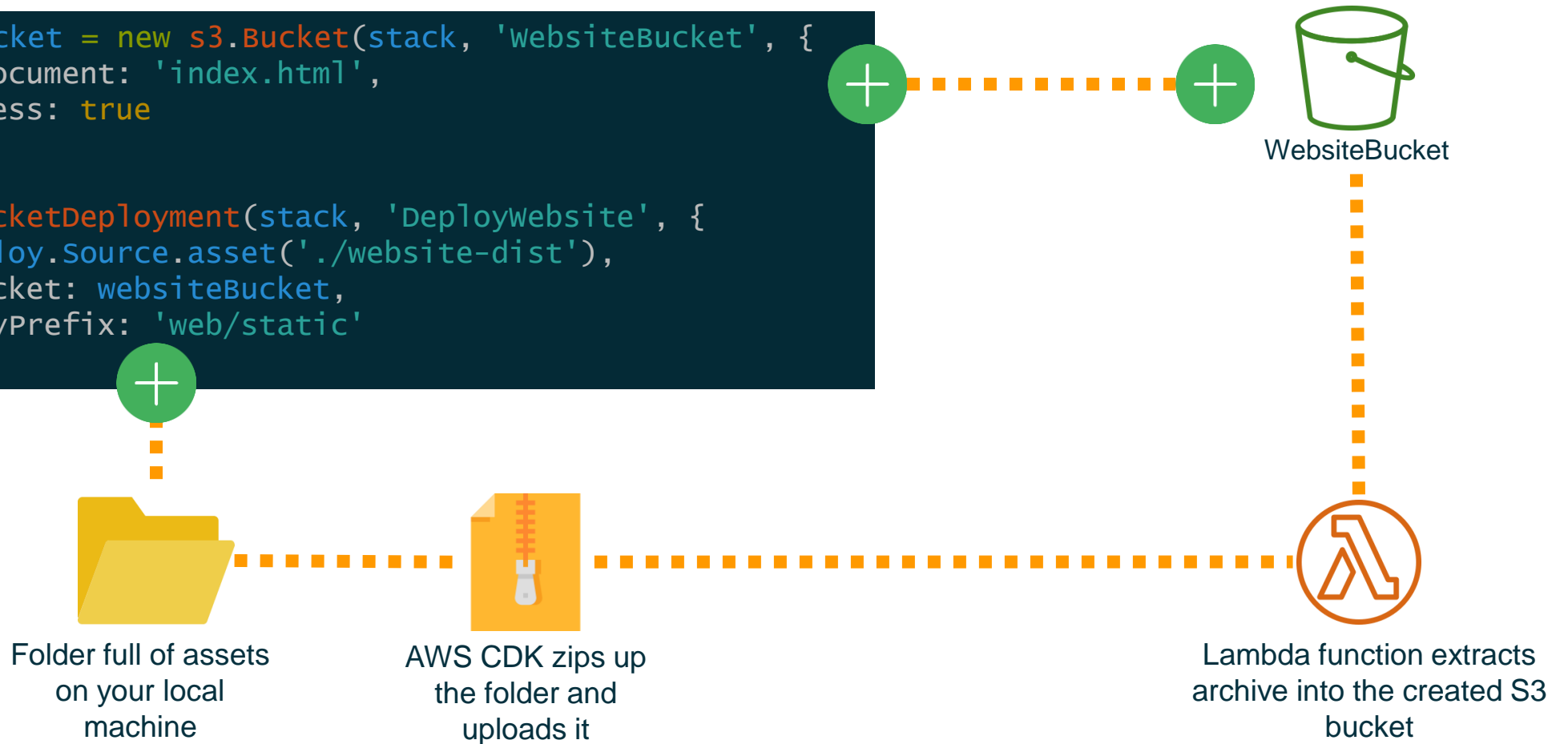
```

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: MyVpcPrivateSubnet2RouteTableECDECEEC
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: MyVpcPrivateSubnet2RouteTableECDECEEC
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

```

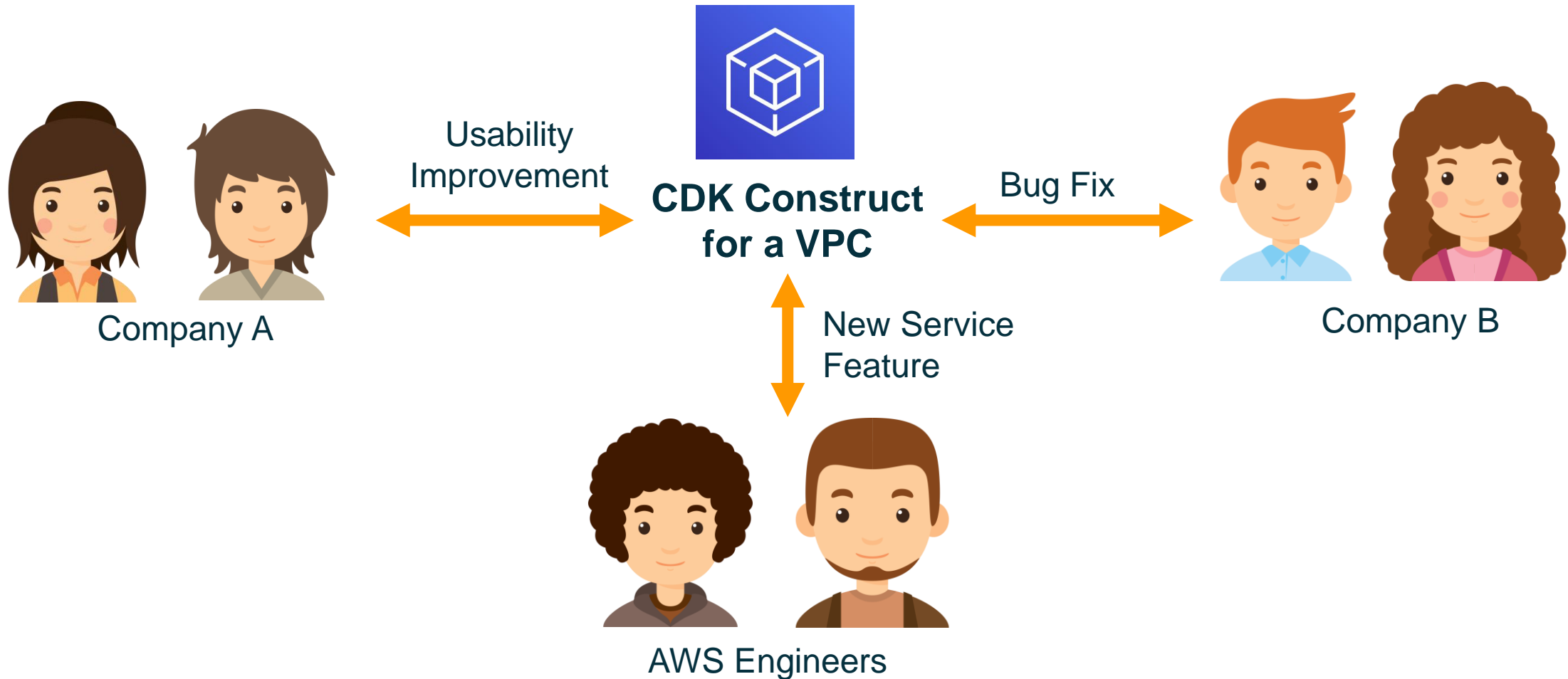
CDK helps with your local workflow too

```
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

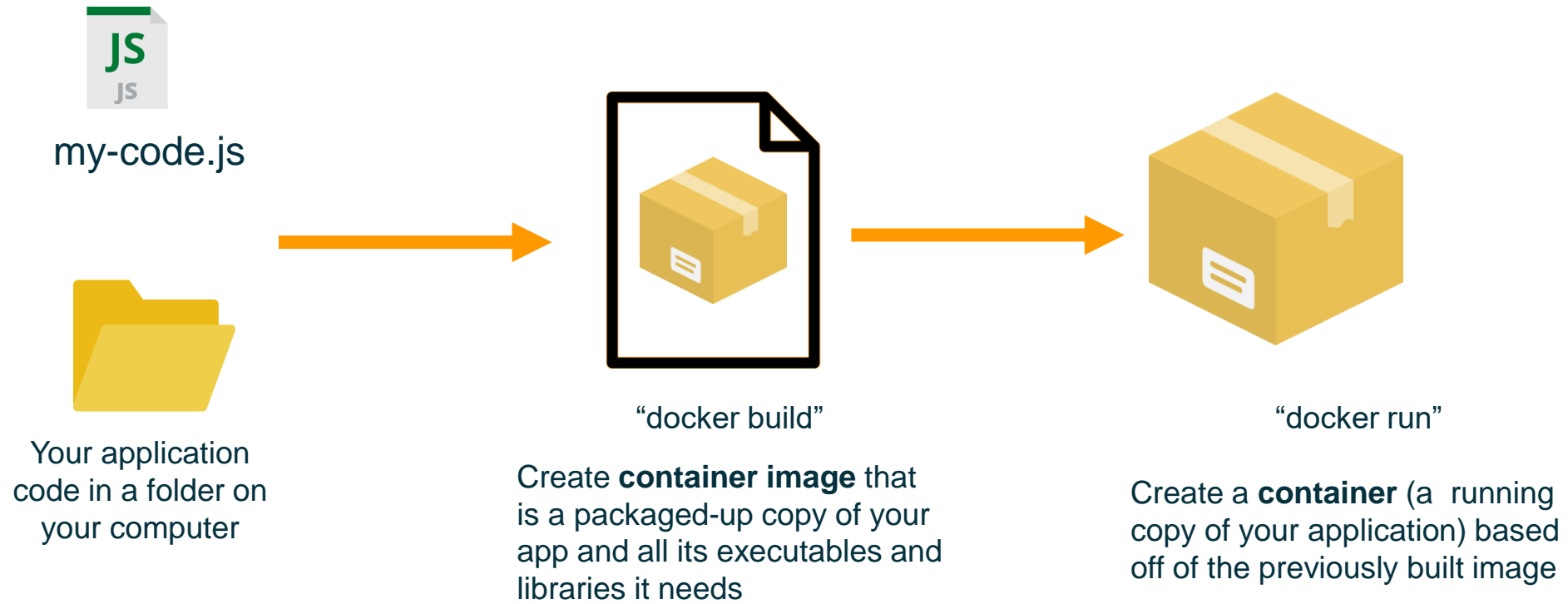


Pros

- 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

First some basic container concepts



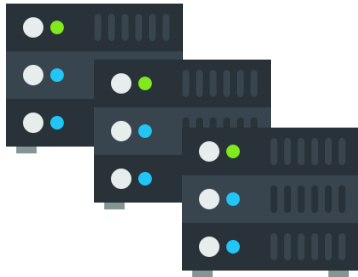
Add some container orchestration concepts

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



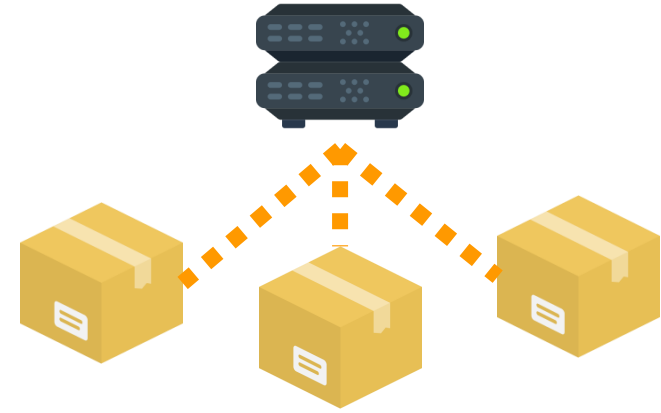
2. Create cluster

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

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

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")
});
```

- 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
});

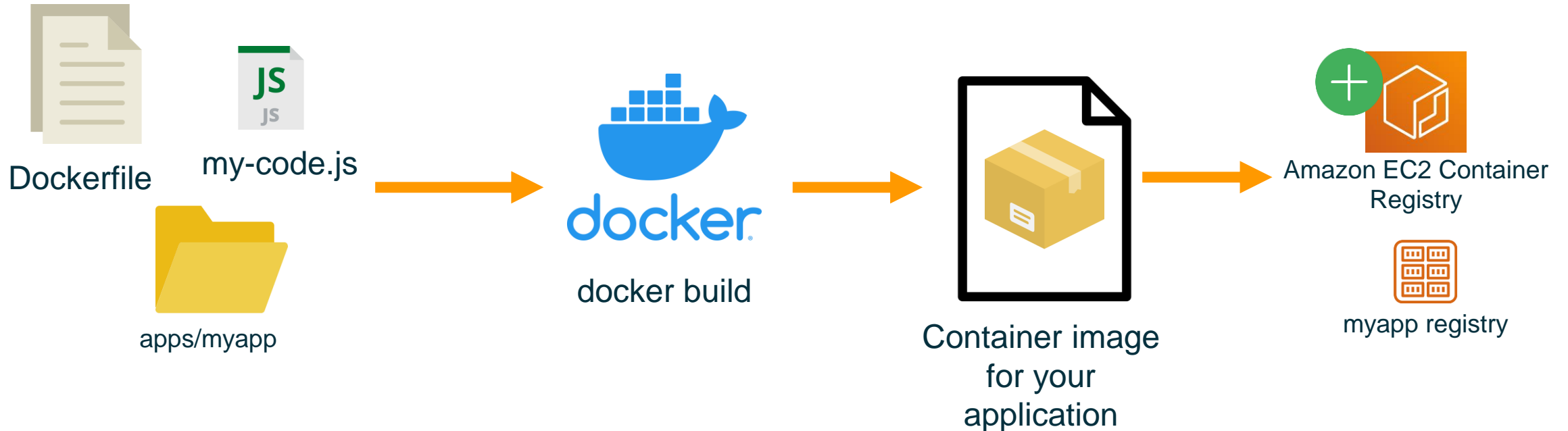
const listener = lb.addListener('PublicListener', { port: 80, open: true });

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),
  }
});
```

@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
});
```

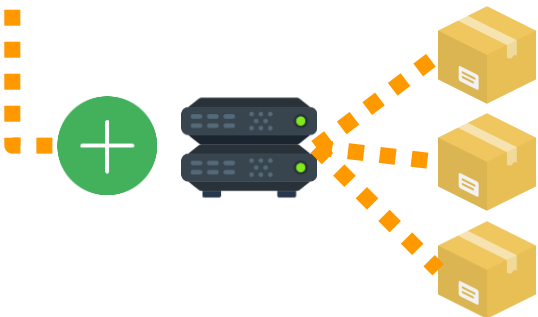
@aws-cdk/aws-ecs-patterns: Load balanced service

```
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 });

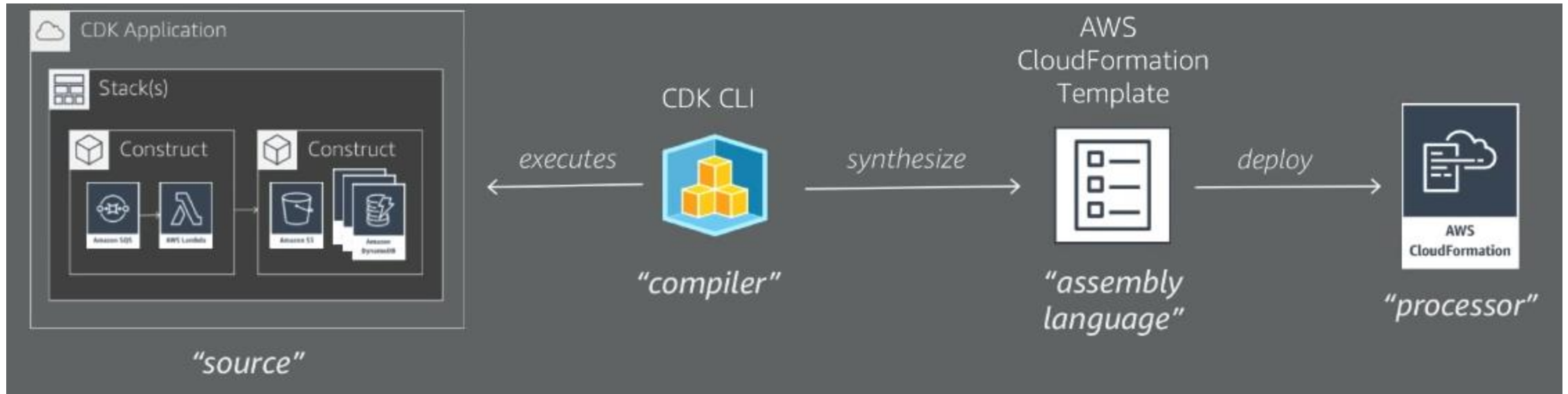
const myService = new ecs_patterns.ApplicationLoadBalancedFargateService(stack, "my-service", {
  cluster,
  desiredCount: 3,
  image: ecs.ContainerImage.fromAsset("apps/myapp")
});
```

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



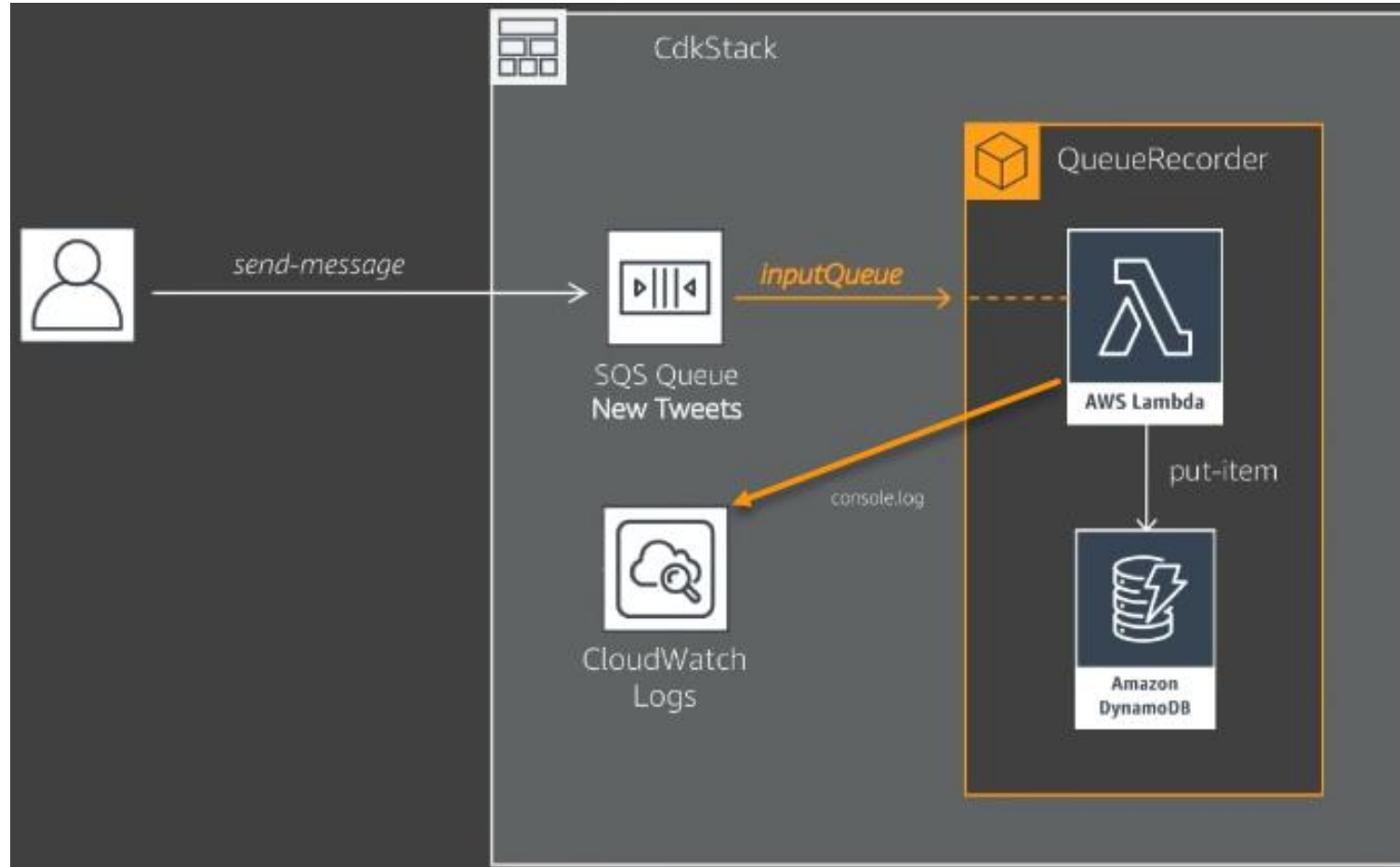
With a few lines we are automatically **building** a Docker container locally, **pushing** it up to the cloud in an Amazon Elastic Container Registry, then **launching** running three copies of it in AWS Fargate, **behind** a load balancer that distributes traffic across all three.

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.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),
  }
});
```

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,
});

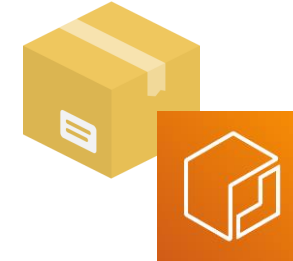
// Grant this task role access to use other resources
myDynamodbTable.grantReadWriteData(taskDefinition.taskRole);
mySnsTopic.grantPublish(taskDefinition.taskRole);
```

- 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 constructs”

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>

Thanks a lot!

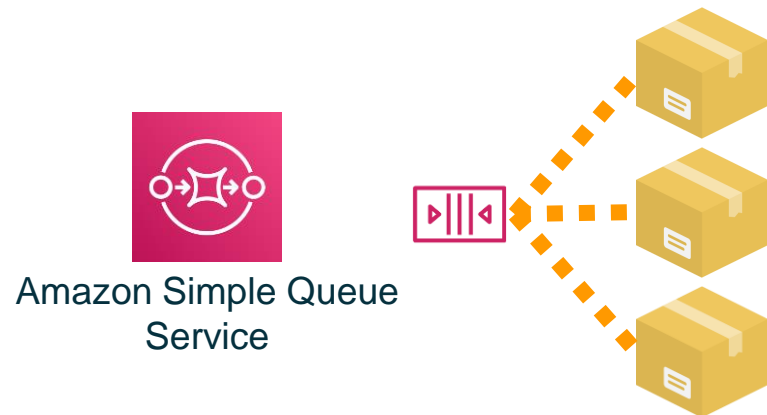
Q & A

Backup

@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!