

Migrating hybrid infra architecture from CloudFormation to AWS CDK

Markus Lindqvist
Senior Solutions Architect
AWS User Group Tampere, November Meetup 2019

Contents

1. The Cloud is a lie
2. Migration from CloudFormation to AWS CDK
3. Least privilege IAM roles
4. Automated container management
5. Conclusions
6. Discussion, Q&A



@markuslindqv

The Cloud is a lie!

Resources:

VirtualMachine:

Type: 'AWS::EC2::Instance'

CreationPolicy:

ResourceSignal:

Timeout: PT10M

Properties:

UserData:

```
'Fn::Base64': !Sub |
  #!/bin/bash -ex
  trap '/opt/aws/bin/cfn-signal --exit-code 0' EXIT
  # run some commands
  /opt/aws/bin/cfn-signal --exit-code 0
```



My journey to AWS

What is CloudFormation

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



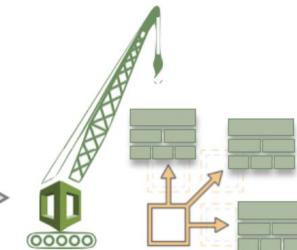
Code your infrastructure from scratch with the CloudFormation template language, in either YAML or JSON format, or start from many available sample templates

IT'S JUST CODE

Codifying your infrastructure allows you to treat your infrastructure as just code. You can author it with any code editor, check it into a version control system, and review the files with team members before deploying into production.

into an S3 bucket

to create a stack based on your template code



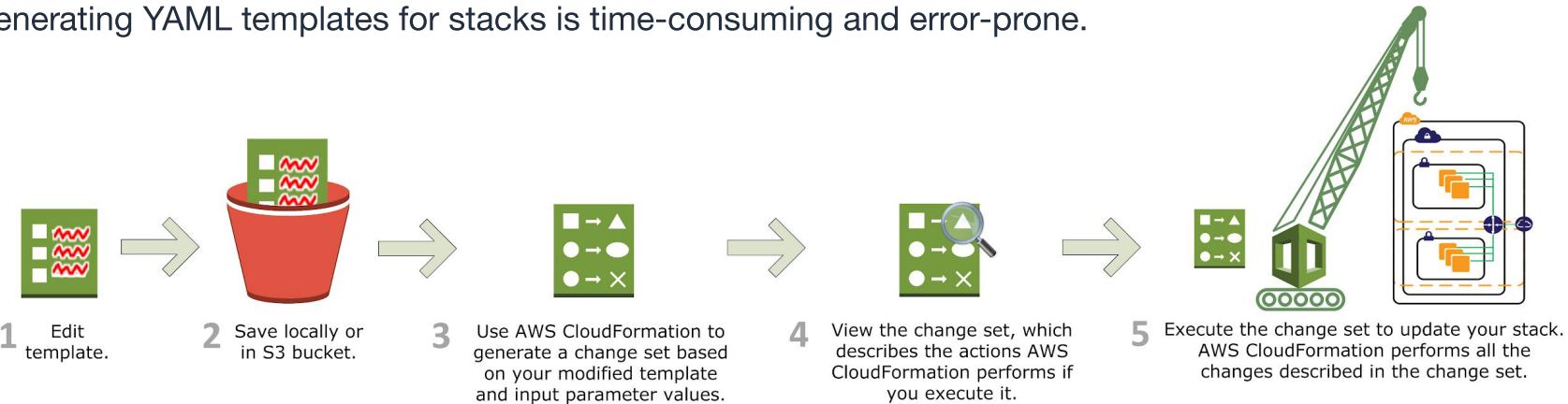
AWS CloudFormation provisions and configures the stacks and resources you specified on your template

CloudFormation workflow

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

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

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



CloudFormation “code”

```

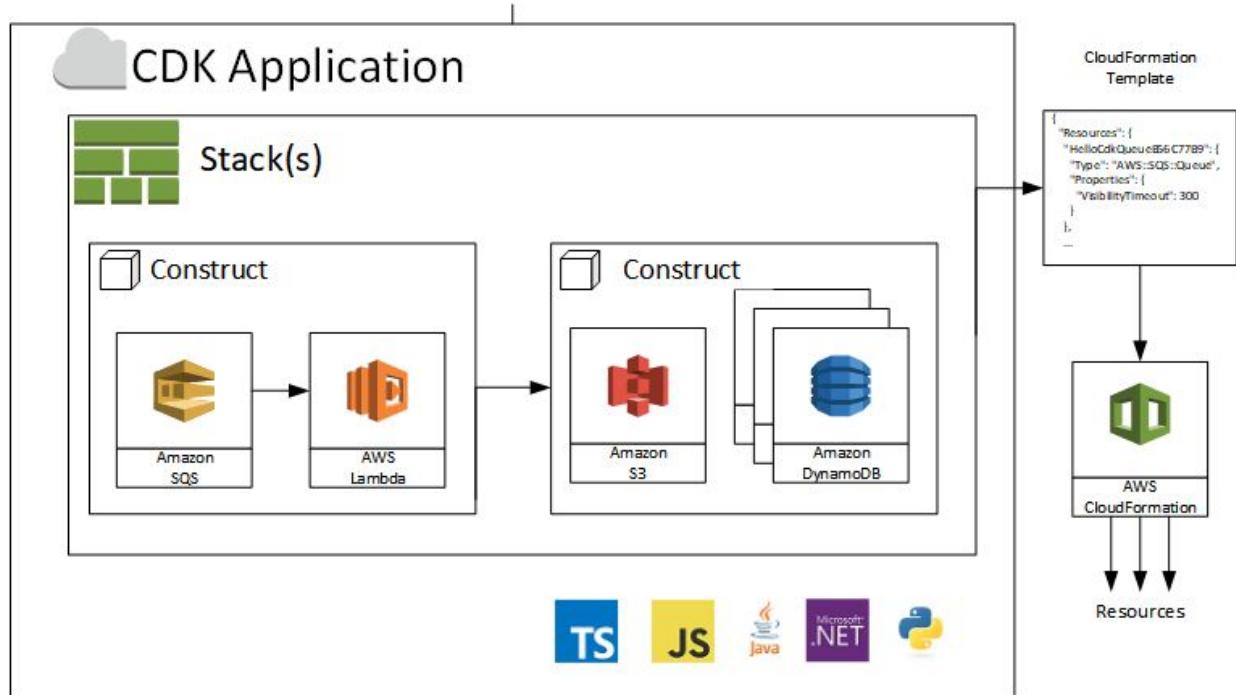
1 Resources:
2   AppNameVpcS226D0497:
3     Type: AWS::EC2::VPC
4     Properties:
5       CidrBlock: 10.0.0.0/16
6       EnableDnsHostnames: true
7       EnableDnsSupport: true
8       InstanceTenancy: default
9     Tags:
10    - Key: Name
11      Value: APROJECT
12    - Key: Application
13      Value: starter-app
14    - Key: CostCenter
15      Value: "10001"
16    - Key: WorkOrder
17      Value: APROJECT
18   Metadata:
19     aws:cdk:path: AppName/AppNameVpc/Resource
20   AppNameVpcPublicSubnet1SubnetC57150B3:
21     Type: AWS::EC2::Subnet
22     Properties:
23       CidrBlock: 10.0.0.0/18
24     VpcId:
25       Ref: AppNameVpcS226D0497
26     AvailabilityZone: eu-west-1a
27     MapPublicIpOnLaunch: true
28   Tags:
29    - Key: Name
30      Value: AppName/AppNameVpc/PublicSubnet1
31    - Key: Application
32      Value: starter-app
33    - Key: CostCenter
34      Value: "10001"
35    - Key: WorkOrder
36      Value: APROJECT
37    - Key: aws-cdk:subnet-name
38      Value: Public
39    - Key: aws-cdk:subnet-type
40      Value: Public
41   Metadata:
42     aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/Subnet
43   AppNameVpcPublicSubnet1RouteTable06FBB427:
44     Type: AWS::EC2::RouteTable
45     Properties:
46       VpcId:
47         Ref: AppNameVpcS226D0497
48   Tags:
49    - Key: Name
50      Value: AppName/AppNameVpc/PublicSubnet1
51    - Key: Application
52      Value: starter-app
53    - Key: CostCenter
54      Value: "10001"
55    - Key: WorkOrder
56      Value: APROJECT
57   Metadata:
58     aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/RouteTable
59   AppNameVpcPublicSubnet1RouteTableAssociation4E017AF2:
60     Type: AWS::EC2::SubnetRouteTableAssociation
61     Properties:
62       RouteTableId:
63         Ref: AppNameVpcPublicSubnet1RouteTable06FBB427
64       SubnetId:
65         Ref: AppNameVpcPublicSubnet1SubnetC57150B3
66     Metadata:
67       aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/RouteTableAssoc
68   AppNameVpcPublicSubnet1DefaultRoute0B5C2D2E:
69     Type: AWS::EC2::Route
70     Properties:
71       RouteTableId:
72         Ref: AppNameVpcPublicSubnet1RouteTable06FBB427
73       DestinationCidrBlock: 0.0.0.0/0
74     Metadata:
75       aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/Subnet
76   AppNameVpcPublicSubnet1RouteTableA90A3EBA:
77     Type: AWS::EC2::RouteTable
78     Properties:
79       VpcId:
80         Ref: AppNameVpcCIGW5506E52
81     Metadata:
82       aws:cdk:path: AppNameVpcRouteTable7A8F97790
83     Tags:
84    - Key: Name
85      Value: AppName/AppNameVpc/PublicSubnet1/DefaultRoute
86   AppNameVpcPublicSubnet1RouteTableE1P77AF6727:
87     Type: AWS::EC2::EIP
88     Properties:
89       Domain: vpc
90     Metadata:
91       aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/EIP
92   AppNameVpcPublicSubnet1NATGateway5D381D31:
93     Type: AWS::EC2::NatGateway
94     Properties:
95       AllocationId:
96         Fn::GetAtt:
97          - Ref: AppNameVpcPublicSubnet1EIP77AF6727
98        - AllocationId
99     SubnetId:
100    Ref: AppNameVpcPublicSubnet1SubnetC57150B3
101   Tags:
102    - Key: Name
103      Value: APROJECT
104   Metadata:
105     aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/NATGateway
106   AppNameVpcPublicSubnet25Subnet72A8F97F:
107     Type: AWS::EC2::Subnet
108     Properties:
109       CidrBlock: 10.0.64.0/18
110   Value: APROJECT
111   Metadata:
112     aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/RouteTable
113   AppNameVpcPublicSubnet1RouteTableAssociation4E017AF2:
114     Type: AWS::EC2::SubnetRouteTableAssociation
115     Properties:
116       RouteTableId:
117         Ref: AppNameVpcPublicSubnet1RouteTable06FBB427
118       SubnetId:
119         Ref: AppNameVpcPublicSubnet1SubnetC57150B3
120     Metadata:
121       aws:cdk:path: AppName/AppNameVpc/PublicSubnet1/RouteTableAssoc
122   AppNameVpcPublicSubnet1DefaultRoute0B5C2D2E:
123     Type: AWS::EC2::Route
124     Properties:
125       RouteTableId:
126         Ref: AppNameVpcPublicSubnet1RouteTable06FBB427
127       DestinationCidrBlock: 0.0.0.0/0
128     Metadata:
129       aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/Subnet
130   AppNameVpcPublicSubnet2RouteTableA90A3EBA:
131     Type: AWS::EC2::RouteTable
132     Properties:
133       VpcId:
134         Ref: AppNameVpcS226D0497
135     Metadata:
136       aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/Subnet
137   AppNameVpcPublicSubnet2RouteTableA90A3EBA:
138     Type: AWS::EC2::RouteTable
139     Properties:
140       VpcId:
141     Metadata:
142       aws:cdk:path: AppNameVpcRouteTable7A8F97790
143     Tags:
144    - Key: Name
145      Value: AppName/AppNameVpc/PublicSubnet2
146    - Key: Application
147      Value: starter-app
148    - Key: CostCenter
149      Value: "10001"
150    - Key: WorkOrder
151      Value: APROJECT
152   Metadata:
153    - Key: Name
154      Value: "10001"
155    - Key: Application
156      Value: APROJECT
157   Metadata:
158     aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/RouteTable
159   AppNameVpcPublicSubnet2RouteTableAssociation5960CAC0:
160     Type: AWS::EC2::SubnetRouteTableAssociation
161     Properties:
162       RouteTableId:
163         Ref: AppNameVpcPublicSubnet2RouteTableA90A3EBA
164       SubnetId:
165         Ref: AppNameVpcPublicSubnet2Subnet72A8F97F
166     Metadata:
167    - Key: Name
168      Value: AppName/AppNameVpc/PublicSubnet2/RouteTableAssociation
169    - Key: Application
170      Value: starter-app
171    - Key: CostCenter
172      Value: "10001"
173    - Key: WorkOrder
174      Value: APROJECT
175    - Key: aws-cdk:subnet-name
176      Value: Public
177    - Key: aws-cdk:subnet-type
178      Value: Public
179   Metadata:
180     aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/Subnet
181   AppNameVpcPublicSubnet2RouteTable7A8F97F:
182     Type: AWS::EC2::RouteTable
183     Properties:
184       VpcId:
185         Ref: AppNameVpcS226D0497
186     Metadata:
187       aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/Subnet
188   AppNameVpcPublicSubnet2RouteTableA90A3EBA:
189     Type: AWS::EC2::RouteTable
190     Properties:
191       VpcId:
192     Metadata:
193       aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/Subnet
194   AppNameVpcPublicSubnet2RouteTableA90A3EBA:
195     Type: AWS::EC2::RouteTable
196     Properties:
197       VpcId:
198     Metadata:
199       aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/Subnet
200   AppNameVpcPublicSubnet2RouteTableAssociation5960CAC0:
201     Type: AWS::EC2::SubnetRouteTableAssociation
202     Properties:
203       RouteTableId:
204         Ref: AppNameVpcPublicSubnet2RouteTableA90A3EBA
205       SubnetId:
206         Ref: AppNameVpcPublicSubnet2Subnet72A8F97F
207     Metadata:
208    - Key: Name
209      Value: AppName/AppNameVpc/PublicSubnet2/RouteTableAssociation
210    - Key: Application
211      Value: starter-app
212    - Key: CostCenter
213      Value: "10001"
214    - Key: WorkOrder
215      Value: APROJECT
216    - Key: aws-cdk:subnet-name
217      Value: "10001"
218   Metadata:
219     aws:cdk:path: AppName/AppNameVpc/PublicSubnet2/NATGateway
220   AppNameVpcPrivateSubnet1SubnetIACA485F:
221     Type: AWS::EC2::Subnet
222     Properties:
223       CidrBlock: 10.0.128.0/18
224     VpcId:
225       Ref: AppNameVpcS226D0497
226     AvailabilityZone: eu-west-1a
227     MapPublicIpOnLaunch: false
228   Tags:
229    - Key: Name
230      Value: AppName/AppNameVpc/PrivateSubnet1
231    - Key: Application
232      Value: starter-app
233    - Key: CostCenter
234      Value: "10001"
235    - Key: WorkOrder
236      Value: APROJECT
237    - Key: aws-cdk:subnet-name
238      Value: "10001"
239   Metadata:
240     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
241   AppNameVpcPrivateSubnet1SubnetIACA485F:
242     Type: AWS::EC2::Subnet
243     Properties:
244       CidrBlock: 10.0.128.0/18
245     VpcId:
246       Ref: AppNameVpcS226D0497
247     AvailabilityZone: eu-west-1a
248     MapPublicIpOnLaunch: false
249   Tags:
250    - Key: Name
251      Value: AppName/AppNameVpc/PrivateSubnet1
252    - Key: Application
253      Value: starter-app
254    - Key: CostCenter
255      Value: "10001"
256    - Key: WorkOrder
257      Value: APROJECT
258    - Key: aws-cdk:subnet-name
259      Value: "10001"
260   Metadata:
261     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
262   AppNameVpcPrivateSubnet1SubnetIACA485F:
263     Type: AWS::EC2::Subnet
264     Properties:
265       CidrBlock: 10.0.128.0/18
266     VpcId:
267       Ref: AppNameVpcS226D0497
268     AvailabilityZone: eu-west-1a
269     MapPublicIpOnLaunch: false
270   Tags:
271    - Key: Name
272      Value: AppName/AppNameVpc/PrivateSubnet1
273    - Key: Application
274      Value: starter-app
275    - Key: CostCenter
276      Value: "10001"
277    - Key: WorkOrder
278      Value: APROJECT
279    - Key: aws-cdk:subnet-name
280      Value: "10001"
281   Metadata:
282     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
283   AppNameVpcPrivateSubnet1SubnetIACA485F:
284     Type: AWS::EC2::Subnet
285     Properties:
286       CidrBlock: 10.0.128.0/18
287     VpcId:
288       Ref: AppNameVpcS226D0497
289     AvailabilityZone: eu-west-1a
290     MapPublicIpOnLaunch: false
291   Tags:
292    - Key: Name
293      Value: AppName/AppNameVpc/PrivateSubnet1
294    - Key: Application
295      Value: starter-app
296    - Key: CostCenter
297      Value: "10001"
298    - Key: WorkOrder
299      Value: APROJECT
300    - Key: aws-cdk:subnet-name
301      Value: "10001"
302   Metadata:
303     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
304   AppNameVpcPrivateSubnet1SubnetIACA485F:
305     Type: AWS::EC2::Subnet
306     Properties:
307       CidrBlock: 10.0.128.0/18
308     VpcId:
309       Ref: AppNameVpcS226D0497
310     AvailabilityZone: eu-west-1a
311     MapPublicIpOnLaunch: false
312   Tags:
313    - Key: Name
314      Value: AppName/AppNameVpc/PrivateSubnet1
315    - Key: Application
316      Value: starter-app
317    - Key: CostCenter
318      Value: "10001"
319    - Key: WorkOrder
320      Value: APROJECT
321    - Key: aws-cdk:subnet-name
322      Value: "10001"
323   Metadata:
324     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
325   AppNameVpcPrivateSubnet1SubnetIACA485F:
326     Type: AWS::EC2::Subnet
327     Properties:
328       CidrBlock: 10.0.128.0/18
329     VpcId:
330       Ref: AppNameVpcS226D0497
331     AvailabilityZone: eu-west-1a
332     MapPublicIpOnLaunch: false
333   Tags:
334    - Key: Name
335      Value: AppName/AppNameVpc/PrivateSubnet1
336    - Key: Application
337      Value: starter-app
338    - Key: CostCenter
339      Value: "10001"
340    - Key: WorkOrder
341      Value: APROJECT
342    - Key: aws-cdk:subnet-name
343      Value: "10001"
344   Metadata:
345     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
346   AppNameVpcPrivateSubnet1SubnetIACA485F:
347     Type: AWS::EC2::Subnet
348     Properties:
349       CidrBlock: 10.0.128.0/18
350     VpcId:
351       Ref: AppNameVpcS226D0497
352     AvailabilityZone: eu-west-1a
353     MapPublicIpOnLaunch: false
354   Tags:
355    - Key: Name
356      Value: AppName/AppNameVpc/PrivateSubnet1
357    - Key: Application
358      Value: starter-app
359    - Key: CostCenter
360      Value: "10001"
361    - Key: WorkOrder
362      Value: APROJECT
363    - Key: aws-cdk:subnet-name
364      Value: "10001"
365   Metadata:
366     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
367   AppNameVpcPrivateSubnet1SubnetIACA485F:
368     Type: AWS::EC2::Subnet
369     Properties:
370       CidrBlock: 10.0.128.0/18
371     VpcId:
372       Ref: AppNameVpcS226D0497
373     AvailabilityZone: eu-west-1a
374     MapPublicIpOnLaunch: false
375   Tags:
376    - Key: Name
377      Value: AppName/AppNameVpc/PrivateSubnet1
378    - Key: Application
379      Value: starter-app
380    - Key: CostCenter
381      Value: "10001"
382    - Key: WorkOrder
383      Value: APROJECT
384    - Key: aws-cdk:subnet-name
385      Value: "10001"
386   Metadata:
387     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
388   AppNameVpcPrivateSubnet1SubnetIACA485F:
389     Type: AWS::EC2::Subnet
390     Properties:
391       CidrBlock: 10.0.128.0/18
392     VpcId:
393       Ref: AppNameVpcS226D0497
394     AvailabilityZone: eu-west-1a
395     MapPublicIpOnLaunch: false
396   Tags:
397    - Key: Name
398      Value: AppName/AppNameVpc/PrivateSubnet1
399    - Key: Application
400      Value: starter-app
401    - Key: CostCenter
402      Value: "10001"
403    - Key: WorkOrder
404      Value: APROJECT
405    - Key: aws-cdk:subnet-name
406      Value: "10001"
407   Metadata:
408     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
409   AppNameVpcPrivateSubnet1SubnetIACA485F:
410     Type: AWS::EC2::Subnet
411     Properties:
412       CidrBlock: 10.0.128.0/18
413     VpcId:
414       Ref: AppNameVpcS226D0497
415     AvailabilityZone: eu-west-1a
416     MapPublicIpOnLaunch: false
417   Tags:
418    - Key: Name
419      Value: AppName/AppNameVpc/PrivateSubnet1
420    - Key: Application
421      Value: starter-app
422    - Key: CostCenter
423      Value: "10001"
424    - Key: WorkOrder
425      Value: APROJECT
426    - Key: aws-cdk:subnet-name
427      Value: "10001"
428   Metadata:
429     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
430   AppNameVpcPrivateSubnet1SubnetIACA485F:
431     Type: AWS::EC2::Subnet
432     Properties:
433       CidrBlock: 10.0.128.0/18
434     VpcId:
435       Ref: AppNameVpcS226D0497
436     AvailabilityZone: eu-west-1a
437     MapPublicIpOnLaunch: false
438   Tags:
439    - Key: Name
440      Value: AppName/AppNameVpc/PrivateSubnet1
441    - Key: Application
442      Value: starter-app
443    - Key: CostCenter
444      Value: "10001"
445    - Key: WorkOrder
446      Value: APROJECT
447    - Key: aws-cdk:subnet-name
448      Value: "10001"
449   Metadata:
450     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
451   AppNameVpcPrivateSubnet1SubnetIACA485F:
452     Type: AWS::EC2::Subnet
453     Properties:
454       CidrBlock: 10.0.128.0/18
455     VpcId:
456       Ref: AppNameVpcS226D0497
457     AvailabilityZone: eu-west-1a
458     MapPublicIpOnLaunch: false
459   Tags:
460    - Key: Name
461      Value: AppName/AppNameVpc/PrivateSubnet1
462    - Key: Application
463      Value: starter-app
464    - Key: CostCenter
465      Value: "10001"
466    - Key: WorkOrder
467      Value: APROJECT
468    - Key: aws-cdk:subnet-name
469      Value: "10001"
470   Metadata:
471     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
472   AppNameVpcPrivateSubnet1SubnetIACA485F:
473     Type: AWS::EC2::Subnet
474     Properties:
475       CidrBlock: 10.0.128.0/18
476     VpcId:
477       Ref: AppNameVpcS226D0497
478     AvailabilityZone: eu-west-1a
479     MapPublicIpOnLaunch: false
480   Tags:
481    - Key: Name
482      Value: AppName/AppNameVpc/PrivateSubnet1
483    - Key: Application
484      Value: starter-app
485    - Key: CostCenter
486      Value: "10001"
487    - Key: WorkOrder
488      Value: APROJECT
489    - Key: aws-cdk:subnet-name
490      Value: "10001"
491   Metadata:
492     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
493   AppNameVpcPrivateSubnet1SubnetIACA485F:
494     Type: AWS::EC2::Subnet
495     Properties:
496       CidrBlock: 10.0.128.0/18
497     VpcId:
498       Ref: AppNameVpcS226D0497
499     AvailabilityZone: eu-west-1a
500     MapPublicIpOnLaunch: false
501   Tags:
502    - Key: Name
503      Value: AppName/AppNameVpc/PrivateSubnet1
504    - Key: Application
505      Value: starter-app
506    - Key: CostCenter
507      Value: "10001"
508    - Key: WorkOrder
509      Value: APROJECT
510    - Key: aws-cdk:subnet-name
511      Value: "10001"
512   Metadata:
513     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
514   AppNameVpcPrivateSubnet1SubnetIACA485F:
515     Type: AWS::EC2::Subnet
516     Properties:
517       CidrBlock: 10.0.128.0/18
518     VpcId:
519       Ref: AppNameVpcS226D0497
520     AvailabilityZone: eu-west-1a
521     MapPublicIpOnLaunch: false
522   Tags:
523    - Key: Name
524      Value: AppName/AppNameVpc/PrivateSubnet1
525    - Key: Application
526      Value: starter-app
527    - Key: CostCenter
528      Value: "10001"
529    - Key: WorkOrder
530      Value: APROJECT
531    - Key: aws-cdk:subnet-name
532      Value: "10001"
533   Metadata:
534     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
535   AppNameVpcPrivateSubnet1SubnetIACA485F:
536     Type: AWS::EC2::Subnet
537     Properties:
538       CidrBlock: 10.0.128.0/18
539     VpcId:
540       Ref: AppNameVpcS226D0497
541     AvailabilityZone: eu-west-1a
542     MapPublicIpOnLaunch: false
543   Tags:
544    - Key: Name
545      Value: AppName/AppNameVpc/PrivateSubnet1
546    - Key: Application
547      Value: starter-app
548    - Key: CostCenter
549      Value: "10001"
550    - Key: WorkOrder
551      Value: APROJECT
552    - Key: aws-cdk:subnet-name
553      Value: "10001"
554   Metadata:
555     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
556   AppNameVpcPrivateSubnet1SubnetIACA485F:
557     Type: AWS::EC2::Subnet
558     Properties:
559       CidrBlock: 10.0.128.0/18
560     VpcId:
561       Ref: AppNameVpcS226D0497
562     AvailabilityZone: eu-west-1a
563     MapPublicIpOnLaunch: false
564   Tags:
565    - Key: Name
566      Value: AppName/AppNameVpc/PrivateSubnet1
567    - Key: Application
568      Value: starter-app
569    - Key: CostCenter
570      Value: "10001"
571    - Key: WorkOrder
572      Value: APROJECT
573    - Key: aws-cdk:subnet-name
574      Value: "10001"
575   Metadata:
576     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
577   AppNameVpcPrivateSubnet1SubnetIACA485F:
578     Type: AWS::EC2::Subnet
579     Properties:
580       CidrBlock: 10.0.128.0/18
581     VpcId:
582       Ref: AppNameVpcS226D0497
583     AvailabilityZone: eu-west-1a
584     MapPublicIpOnLaunch: false
585   Tags:
586    - Key: Name
587      Value: AppName/AppNameVpc/PrivateSubnet1
588    - Key: Application
589      Value: starter-app
590    - Key: CostCenter
591      Value: "10001"
592    - Key: WorkOrder
593      Value: APROJECT
594    - Key: aws-cdk:subnet-name
595      Value: "10001"
596   Metadata:
597     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
598   AppNameVpcPrivateSubnet1SubnetIACA485F:
599     Type: AWS::EC2::Subnet
600     Properties:
601       CidrBlock: 10.0.128.0/18
602     VpcId:
603       Ref: AppNameVpcS226D0497
604     AvailabilityZone: eu-west-1a
605     MapPublicIpOnLaunch: false
606   Tags:
607    - Key: Name
608      Value: AppName/AppNameVpc/PrivateSubnet1
609    - Key: Application
610      Value: starter-app
611    - Key: CostCenter
612      Value: "10001"
613    - Key: WorkOrder
614      Value: APROJECT
615    - Key: aws-cdk:subnet-name
616      Value: "10001"
617   Metadata:
618     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
619   AppNameVpcPrivateSubnet1SubnetIACA485F:
620     Type: AWS::EC2::Subnet
621     Properties:
622       CidrBlock: 10.0.128.0/18
623     VpcId:
624       Ref: AppNameVpcS226D0497
625     AvailabilityZone: eu-west-1a
626     MapPublicIpOnLaunch: false
627   Tags:
628    - Key: Name
629      Value: AppName/AppNameVpc/PrivateSubnet1
630    - Key: Application
631      Value: starter-app
632    - Key: CostCenter
633      Value: "10001"
634    - Key: WorkOrder
635      Value: APROJECT
636    - Key: aws-cdk:subnet-name
637      Value: "10001"
638   Metadata:
639     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
640   AppNameVpcPrivateSubnet1SubnetIACA485F:
641     Type: AWS::EC2::Subnet
642     Properties:
643       CidrBlock: 10.0.128.0/18
644     VpcId:
645       Ref: AppNameVpcS226D0497
646     AvailabilityZone: eu-west-1a
647     MapPublicIpOnLaunch: false
648   Tags:
649    - Key: Name
650      Value: AppName/AppNameVpc/PrivateSubnet1
651    - Key: Application
652      Value: starter-app
653    - Key: CostCenter
654      Value: "10001"
655    - Key: WorkOrder
656      Value: APROJECT
657    - Key: aws-cdk:subnet-name
658      Value: "10001"
659   Metadata:
660     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
661   AppNameVpcPrivateSubnet1SubnetIACA485F:
662     Type: AWS::EC2::Subnet
663     Properties:
664       CidrBlock: 10.0.128.0/18
665     VpcId:
666       Ref: AppNameVpcS226D0497
667     AvailabilityZone: eu-west-1a
668     MapPublicIpOnLaunch: false
669   Tags:
670    - Key: Name
671      Value: AppName/AppNameVpc/PrivateSubnet1
672    - Key: Application
673      Value: starter-app
674    - Key: CostCenter
675      Value: "10001"
676    - Key: WorkOrder
677      Value: APROJECT
678    - Key: aws-cdk:subnet-name
679      Value: "10001"
680   Metadata:
681     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
682   AppNameVpcPrivateSubnet1SubnetIACA485F:
683     Type: AWS::EC2::Subnet
684     Properties:
685       CidrBlock: 10.0.128.0/18
686     VpcId:
687       Ref: AppNameVpcS226D0497
688     AvailabilityZone: eu-west-1a
689     MapPublicIpOnLaunch: false
690   Tags:
691    - Key: Name
692      Value: AppName/AppNameVpc/PrivateSubnet1
693    - Key: Application
694      Value: starter-app
695    - Key: CostCenter
696      Value: "10001"
697    - Key: WorkOrder
698      Value: APROJECT
699    - Key: aws-cdk:subnet-name
700      Value: "10001"
701   Metadata:
702     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
703   AppNameVpcPrivateSubnet1SubnetIACA485F:
704     Type: AWS::EC2::Subnet
705     Properties:
706       CidrBlock: 10.0.128.0/18
707     VpcId:
708       Ref: AppNameVpcS226D0497
709     AvailabilityZone: eu-west-1a
710     MapPublicIpOnLaunch: false
711   Tags:
712    - Key: Name
713      Value: AppName/AppNameVpc/PrivateSubnet1
714    - Key: Application
715      Value: starter-app
716    - Key: CostCenter
717      Value: "10001"
718    - Key: WorkOrder
719      Value: APROJECT
720    - Key: aws-cdk:subnet-name
721      Value: "10001"
722   Metadata:
723     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
724   AppNameVpcPrivateSubnet1SubnetIACA485F:
725     Type: AWS::EC2::Subnet
726     Properties:
727       CidrBlock: 10.0.128.0/18
728     VpcId:
729       Ref: AppNameVpcS226D0497
730     AvailabilityZone: eu-west-1a
731     MapPublicIpOnLaunch: false
732   Tags:
733    - Key: Name
734      Value: AppName/AppNameVpc/PrivateSubnet1
735    - Key: Application
736      Value: starter-app
737    - Key: CostCenter
738      Value: "10001"
739    - Key: WorkOrder
740      Value: APROJECT
741    - Key: aws-cdk:subnet-name
742      Value: "10001"
743   Metadata:
744     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
745   AppNameVpcPrivateSubnet1SubnetIACA485F:
746     Type: AWS::EC2::Subnet
747     Properties:
748       CidrBlock: 10.0.128.0/18
749     VpcId:
750       Ref: AppNameVpcS226D0497
751     AvailabilityZone: eu-west-1a
752     MapPublicIpOnLaunch: false
753   Tags:
754    - Key: Name
755      Value: AppName/AppNameVpc/PrivateSubnet1
756    - Key: Application
757      Value: starter-app
758    - Key: CostCenter
759      Value: "10001"
760    - Key: WorkOrder
761      Value: APROJECT
762    - Key: aws-cdk:subnet-name
763      Value: "10001"
764   Metadata:
765     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
766   AppNameVpcPrivateSubnet1SubnetIACA485F:
767     Type: AWS::EC2::Subnet
768     Properties:
769       CidrBlock: 10.0.128.0/18
770     VpcId:
771       Ref: AppNameVpcS226D0497
772     AvailabilityZone: eu-west-1a
773     MapPublicIpOnLaunch: false
774   Tags:
775    - Key: Name
776      Value: AppName/AppNameVpc/PrivateSubnet1
777    - Key: Application
778      Value: starter-app
779    - Key: CostCenter
780      Value: "10001"
781    - Key: WorkOrder
782      Value: APROJECT
783    - Key: aws-cdk:subnet-name
784      Value: "10001"
785   Metadata:
786     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
787   AppNameVpcPrivateSubnet1SubnetIACA485F:
788     Type: AWS::EC2::Subnet
789     Properties:
790       CidrBlock: 10.0.128.0/18
791     VpcId:
792       Ref: AppNameVpcS226D0497
793     AvailabilityZone: eu-west-1a
794     MapPublicIpOnLaunch: false
795   Tags:
796    - Key: Name
797      Value: AppName/AppNameVpc/PrivateSubnet1
798    - Key: Application
799      Value: starter-app
800    - Key: CostCenter
801      Value: "10001"
802    - Key: WorkOrder
803      Value: APROJECT
804    - Key: aws-cdk:subnet-name
805      Value: "10001"
806   Metadata:
807     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
808   AppNameVpcPrivateSubnet1SubnetIACA485F:
809     Type: AWS::EC2::Subnet
810     Properties:
811       CidrBlock: 10.0.128.0/18
812     VpcId:
813       Ref: AppNameVpcS226D0497
814     AvailabilityZone: eu-west-1a
815     MapPublicIpOnLaunch: false
816   Tags:
817    - Key: Name
818      Value: AppName/AppNameVpc/PrivateSubnet1
819    - Key: Application
820      Value: starter-app
821    - Key: CostCenter
822      Value: "10001"
823    - Key: WorkOrder
824      Value: APROJECT
825    - Key: aws-cdk:subnet-name
826      Value: "10001"
827   Metadata:
828     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
829   AppNameVpcPrivateSubnet1SubnetIACA485F:
830     Type: AWS::EC2::Subnet
831     Properties:
832       CidrBlock: 10.0.128.0/18
833     VpcId:
834       Ref: AppNameVpcS226D0497
835     AvailabilityZone: eu-west-1a
836     MapPublicIpOnLaunch: false
837   Tags:
838    - Key: Name
839      Value: AppName/AppNameVpc/PrivateSubnet1
840    - Key: Application
841      Value: starter-app
842    - Key: CostCenter
843      Value: "10001"
844    - Key: WorkOrder
845      Value: APROJECT
846    - Key: aws-cdk:subnet-name
847      Value: "10001"
848   Metadata:
849     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
850   AppNameVpcPrivateSubnet1SubnetIACA485F:
851     Type: AWS::EC2::Subnet
852     Properties:
853       CidrBlock: 10.0.128.0/18
854     VpcId:
855       Ref: AppNameVpcS226D0497
856     AvailabilityZone: eu-west-1a
857     MapPublicIpOnLaunch: false
858   Tags:
859    - Key: Name
860      Value: AppName/AppNameVpc/PrivateSubnet1
861    - Key: Application
862      Value: starter-app
863    - Key: CostCenter
864      Value: "10001"
865    - Key: WorkOrder
866      Value: APROJECT
867    - Key: aws-cdk:subnet-name
868      Value: "10001"
869   Metadata:
870     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
871   AppNameVpcPrivateSubnet1SubnetIACA485F:
872     Type: AWS::EC2::Subnet
873     Properties:
874       CidrBlock: 10.0.128.0/18
875     VpcId:
876       Ref: AppNameVpcS226D0497
877     AvailabilityZone: eu-west-1a
878     MapPublicIpOnLaunch: false
879   Tags:
880    - Key: Name
881      Value: AppName/AppNameVpc/PrivateSubnet1
882    - Key: Application
883      Value: starter-app
884    - Key: CostCenter
885      Value: "10001"
886    - Key: WorkOrder
887      Value: APROJECT
888    - Key: aws-cdk:subnet-name
889      Value: "10001"
890   Metadata:
891     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
892   AppNameVpcPrivateSubnet1SubnetIACA485F:
893     Type: AWS::EC2::Subnet
894     Properties:
895       CidrBlock: 10.0.128.0/18
896     VpcId:
897       Ref: AppNameVpcS226D0497
898     AvailabilityZone: eu-west-1a
899     MapPublicIpOnLaunch: false
900   Tags:
901    - Key: Name
902      Value: AppName/AppNameVpc/PrivateSubnet1
903    - Key: Application
904      Value: starter-app
905    - Key: CostCenter
906      Value: "10001"
907    - Key: WorkOrder
908      Value: APROJECT
909    - Key: aws-cdk:subnet-name
910      Value: "10001"
911   Metadata:
912     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
913   AppNameVpcPrivateSubnet1SubnetIACA485F:
914     Type: AWS::EC2::Subnet
915     Properties:
916       CidrBlock: 10.0.128.0/18
917     VpcId:
918       Ref: AppNameVpcS226D0497
919     AvailabilityZone: eu-west-1a
920     MapPublicIpOnLaunch: false
921   Tags:
922    - Key: Name
923      Value: AppName/AppNameVpc/PrivateSubnet1
924    - Key: Application
925      Value: starter-app
926    - Key: CostCenter
927      Value: "10001"
928    - Key: WorkOrder
929      Value: APROJECT
930    - Key: aws-cdk:subnet-name
931      Value: "10001"
932   Metadata:
933     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
934   AppNameVpcPrivateSubnet1SubnetIACA485F:
935     Type: AWS::EC2::Subnet
936     Properties:
937       CidrBlock: 10.0.128.0/18
938     VpcId:
939       Ref: AppNameVpcS226D0497
940     AvailabilityZone: eu-west-1a
941     MapPublicIpOnLaunch: false
942   Tags:
943    - Key: Name
944      Value: AppName/AppNameVpc/PrivateSubnet1
945    - Key: Application
946      Value: starter-app
947    - Key: CostCenter
948      Value: "10001"
949    - Key: WorkOrder
950      Value: APROJECT
951    - Key: aws-cdk:subnet-name
952      Value: "10001"
953   Metadata:
954     aws:cdk:path: AppName/AppNameVpc/PrivateSubnet1/Subnet
955   AppNameVpcPrivateSubnet1SubnetIACA485F:
956     Type: AWS::EC2::Subnet
957     Properties:
958       CidrBlock: 10.0.128.0/18
959     VpcId:
960       Ref: AppNameVpcS226D0497
961     AvailabilityZone: eu-west-1a
962     MapPublicIpOnLaunch: false
963   Tags:
964    - Key: Name
965      Value: AppName/AppNameVpc/PrivateSubnet1
966    - Key: Application
967      Value: starter-app
968    - Key: CostCenter
969      Value: "10001"
970    - Key: WorkOrder
971      Value: APROJECT
972   
```

What is AWS CDK?

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

Concepts of CDK:

- Generate CloudFormation templates
- Deployment tools included
- Reusable constructs



Using TypeScript for producing YAML templates

```
import * as cdk from '@aws-cdk/core';
import * as s3 from '@aws-cdk/aws-s3';

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

The diagram illustrates the process of generating a CloudFormation stack from a TypeScript codebase. It starts with a code snippet on the left, which defines an AWS Lambda function and a CloudWatch Log Stream. A large green arrow points from this code to a synthesized CloudFormation template on the right. The template includes resources like a Lambda function, a log group, and a log stream, each with specific properties and metadata. Three callout bubbles provide context for the code:

- All CDK apps have a single 'app'**: Points to the line `const myApp = new cdk.App();`.
- An app can have multiple stacks**: Points to the line `const stack = new cdk.Stack(myApp, 'BucketStack');`.
- Different resources are instantiated in stacks**: Points to the line `new s3.Bucket(stack, 'ExampleBucket', {`.

Annotations on the synthesized template include:

- `DeletionPolicy: Delete`
- `Metadata:`
- `aws:cdk:path: BucketStack/ExampleBucket/Resource`
- `Metadata`
- `.cdk=1.0.0,jsii-runtime=node.js/v12.0.0`

aws-cdk-example.ts

CDK synth

Synthesized CloudFormation stack

Main advantages of AWS CDK for developers

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

The screenshot shows a code editor with AWS CDK code. A tooltip is displayed over the line `service.loadBalancer.`, listing properties and methods available for the `BaseLoadBalancer` class. The tooltip includes:

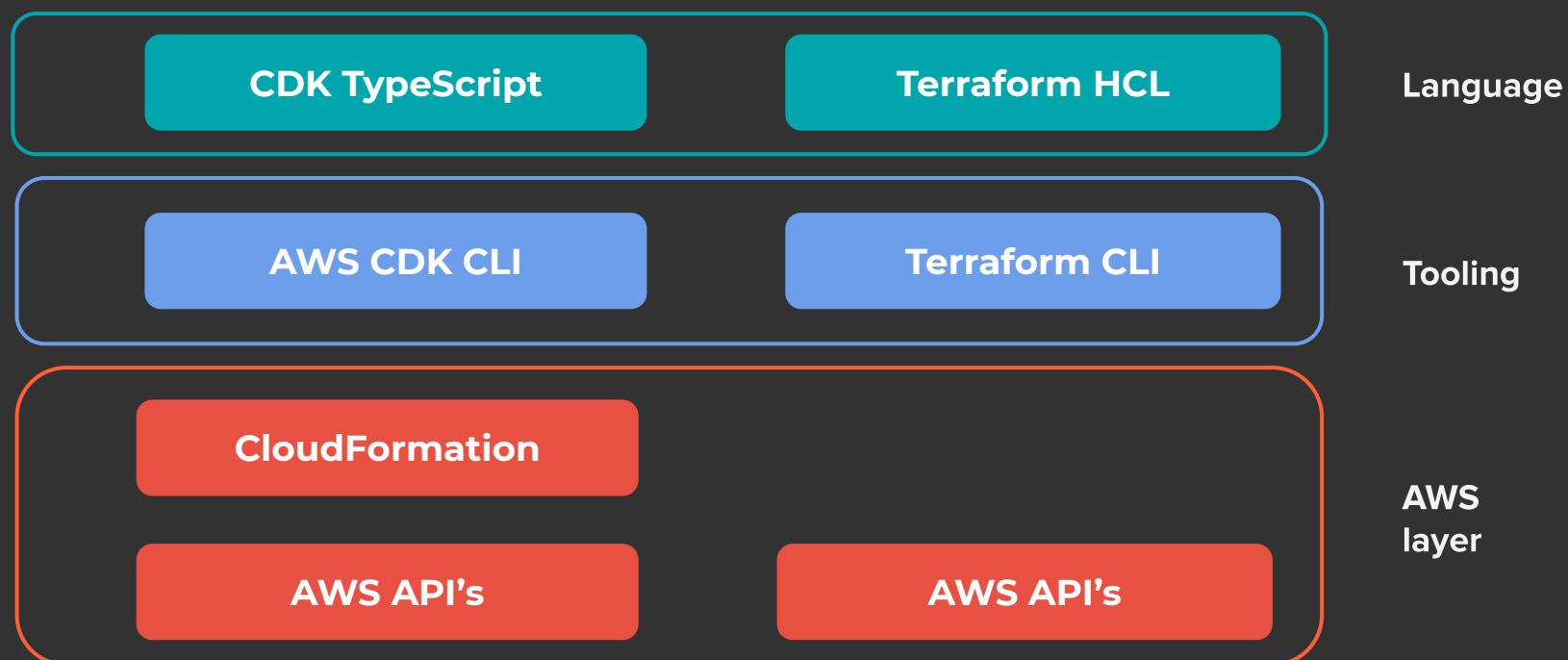
- loadBalancerArn
- loadBalancerCanonicalHostedZoneId
- loadBalancerDnsName
- loadBalancerFullName
- loadBalancerName
- loadBalancerSecurityGroups
- node
- removeAttribute
- setAttribute

Below the tooltip, the `loadBalancerArn` property is described as "The ARN of this load balancer" with an example value: `arn:aws:elasticloadbalancing:us-west-2:123456789012:loadbalancer/app/my-internal-load-balancer/50dc6c495c0c9188`.

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

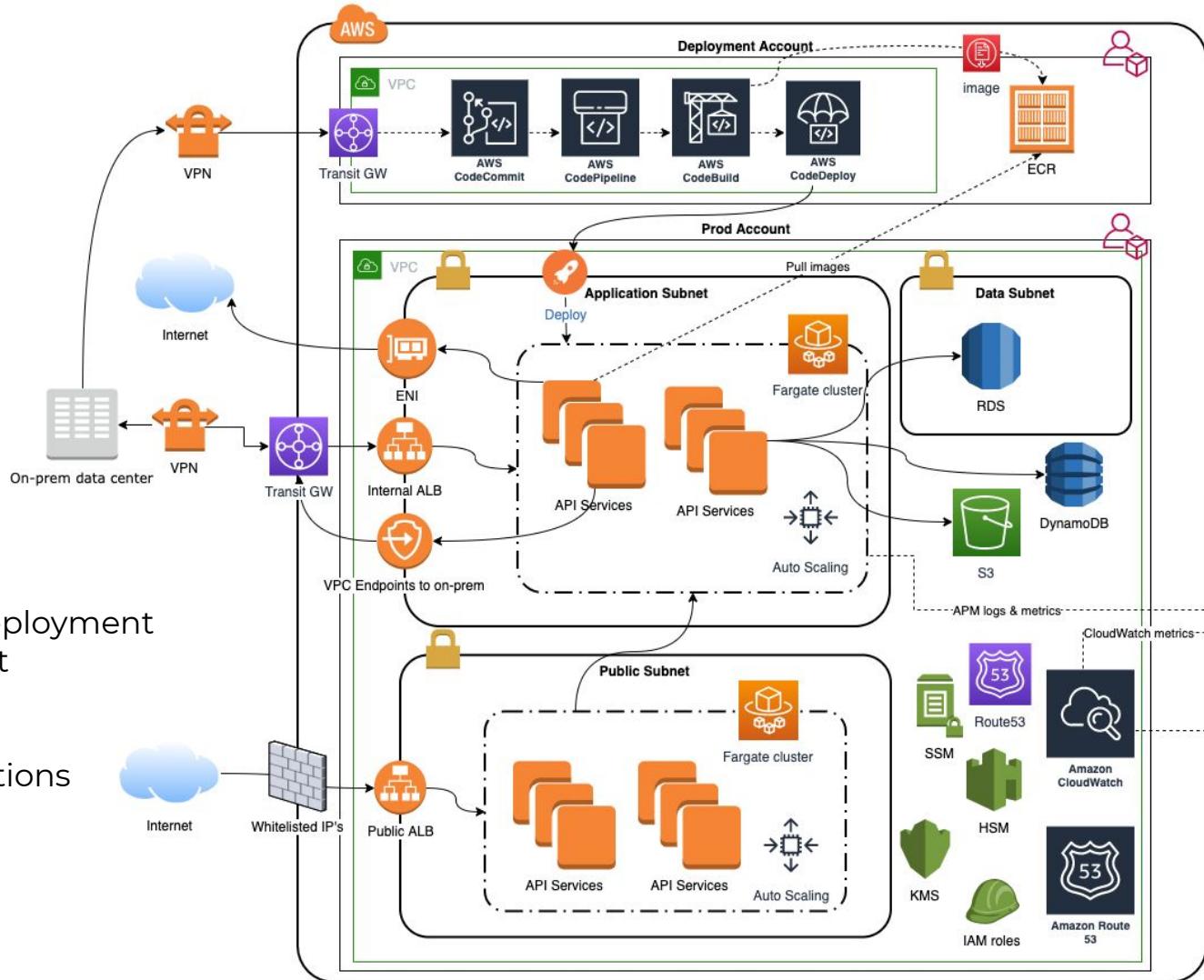
const myApp = new core.App()
new MyFargateConstructStack(myApp)
```

Differences between AWS CDK and Terraform



Migration from CloudFormation to AWS CDK

Production grade AWS hybrid infrastructure



Features:

- Fargate cluster
- Cross Account Deployment
- Log management
- Monitoring
- Backups
- On-prem connections
- High security

The infrastructure to be migrated from CF to CDK

Data stacks to be migrated:

- CodeCommit repositories
- ECR repositories
- S3 bucket stacks
- Relational databases
- DynamoDB databases
- SSM parameters
- Route53 stacks

The databases and repositories need migrations before they can be redeployed as AWS CDK stacks.

Functional stacks to be migrated:

- CodePipeline stacks
- Lambda stacks (devops purposes)
- Fargate clusters with public load balancer
- Fargate clusters with internal load balancer

Functional stacks can be fully redeployed (almost) any time with mostly just DNS changes due to load balancer addresses changing.

Redeploying CodeCommit as AWS CDK

Yes, it really is this simple :-)

```
const repositoryNames = [ 'api-service-1', 'api-service-2', 'api-service-3', 'api-service-4', ... ];

export const createCodeCommitStack = (app: cdk.App, props: cdk.StackProps) => {
  const stack = new cdk.Stack(app, 'code-commit-repositories', props);
  const codeCommitRepos = repositoryNames.map((name) =>
    new codecommit.Repository(stack, ` ${name} `, {
      repositoryName: name,
    }));
  codeCommitRepos.forEach((repo) => applyTags(repo, props.tags as Tags));
};
```

Redeploying ECR's as AWS CDK

```
export const createEcrRepositories = (app: cdk.Construct, props: cdk.StackProps) => {
  const stack = new cdk.Stack(app, 'ecr-repositories', props);
  const repositories = ['repository-1', 'repository-2', 'repository-3'];
  repositories.forEach((r) => createEcrRepository(stack, r));
}

const createEcrRepository = (stack: cdk.Stack, name: string) => {
  const accounts = [DevopsAccountNumber, DevAccountNumber, ProdAccountNumber];
  const ecrRepository = new ecr.Repository(stack, `${name}Ecr`, {
    repositoryName: name,
  });
  ecrRepository.addToResourcePolicy(new iam.PolicyStatement({
    actions: ['ecr:GetDownloadUrlForLayer', 'ecr:BatchGetImage', 'ecr:BatchCheckLayerAvailability'],
    principals: accounts.map((account) => new iam.AccountPrincipal(account)),
  }));
};
```

Least privilege IAM roles

Integrating Datadog monitoring

```
npm install --save-dev  
aws-cdk-datadog-ecs-integration
```

Adding a Datadog sidecar container couldn't be easier. Using pre-made reusable AWS CDK construct libraries makes extending stacks extremely easy.

```
const environment = {  
    DD_APM_IGNORE_RESOURCES: 'GET .*health-check',  
    DD_TAGS: `env:${env}`  
};  
  
new DatadogFargateIntegration(stack, `${service.name}`,  
    taskDefinition,  
    {  
        datadogApiKey,  
        environment,  
        logging,  
    }  
);
```

Log delivery to Kinesis stream

Required for **audit** and **compliance** purposes.

With just few lines, any log group can be streamed to a Kinesis stream on a compliance / logging account which is used as the permanent store.

```
new logs.SubscriptionFilter(stack, `${service.name}SubscriptionFilter`, {  
  logGroup,  
  filterPattern: logs.FilterPattern.allEvents(),  
  destination: new logs_destinations.KinesisDestination(  
    kinesis.Stream.fromStreamArn(stack, `${service.name}Stream`, service.logDestinationArn)),  
});
```

Autoscaling the cluster capacity

```
const scaling = s.autoScaleTaskCount({
    minCapacity: services[i].desiredCount,
    maxCapacity: services[i].maxCount || 2
});
const cpuProps = services[i].cpuScalingProps;
if (cpuProps) {
    scaling.scaleOnCpuUtilization(`CpuScaling`, cpuProps);
}
const memoryProps = services[i].memoryScalingProps;
if (memoryProps) {
    scaling.scaleOnMemoryUtilization(`MemoryScaling`, memoryProps);
}
```

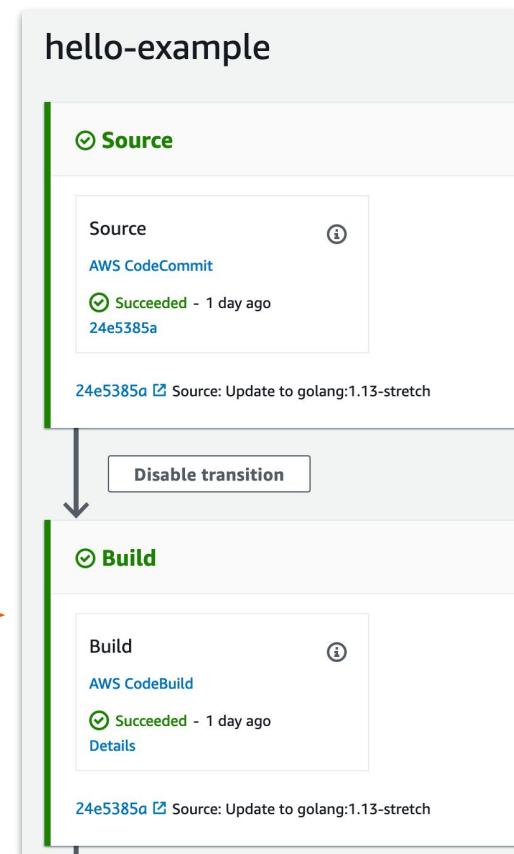
Automated scaling can be configured for each service based on **CPU** or **memory** usage, or as combination of both.

You could also scale by the number of requests.

Building images in AWS

```
const source = new codepipeline.Artifact('Source');
pipeline.addStage({ stageName: 'Source',
  actions: [ new codepipeline_actions.CodeCommitSourceAction({
    actionName: 'Source',
    repository,
    output: source,
    role: codeCommitRole(pipeline.stack, repository.repositoryName),
  })]]);
});
```

```
const buildOutput = new codepipeline.Artifact();
pipeline.addStage({ stageName: 'Build',
  actions: [new codepipeline_actions.CodeBuildAction({
    actionName: 'Build',
    input: source,
    project,
    outputs: [buildOutput],
  })]]);
});
```



Deploying Lambdas with CDK

- Previously we used the [Serverless Framework](#) and [AWS SAM](#) to deploy Lambda functions to perform simple tasks
- These were refactored to AWS CDK as well → less technologies, less complexity
- The code is slightly more verbose than pure serverless configuration
- Inline code makes deployment painless

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

Using unsupported CloudFormation constructs

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

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

Unit testing AWS CDK infrastructure

- Still with experimental status in the API, but enables users to write unit tests against their stacks
- Works just like unit testing any other code
- <https://github.com/aws/aws-cdk/blob/master/packages/@aws-cdk/assert/README.md>

```
'bucket without encryption'(test: Test) {
  const stack = new cdk.Stack();
  new s3.Bucket(stack, 'MyBucket', {
    encryption: s3.BucketEncryption.UNENCRYPTED
  });

  expect(stack).toMatch({
    "Resources": {
      "MyBucketF68F3FF0": {
        "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain",
        "UpdateReplacePolicy": "Retain",
      }
    }
  });
}

test.done();
},
```

Least privilege IAM roles

How to pass your security audits

THE CHALLENGE OF "LEAST-PRIVILEGED" IAM ROLES

- Functions should only be allowed to do what they are tasked with
- AWS IAM model is extremely powerful, yet hard to get right
- Human factor

Effect: Allow

Actions:

- 'dynamodb:*

```
1 # store pdf content in DynamoDB table
2 dynamodb_client.put_item[TableName=TABLE_NAME,
3   Item={"email": {"S": parser['From']},
4         "subject": {"S": parser['Subject']},
5         "received": {"S": str(datetime.utcnow()),
6           "filename": {"S": file_name},
7           "requestid": {"S": context.aws_request
8             "content": {"S": pdf_content.decode("u
```

```
2   name: aws
3   runtime: python3.6
4   timeout: 300
5   memorySize: 3008
6   profile: demo
7   iamRoleStatements:
8     - Effect: Allow
9       Action:
```

- 'dynamodb:*

Still a non-optimal
definition!

14

Least privilege IAM roles

With AWS CDK it's so easy that developers actually do it.

```
bucket.g
bucket.a
action
resource
principal
code
code
new
new

grantDelete
grantPublicAccess
grantPut
grantRead
grantReadWrite
grantWrite
toString
bucketRegionalDomainName
```



Ari Palo
@aripalo

This is why I ❤️ #awscdk & its grant API:

```
const fn = new Function(/*...*/);
const table = new Table(/*...*/);
table.grantReadData(fn.role);
```

Much more intuitive & developer friendly to define granular security vs writing regular IAM policy documents which usually left too open.

Least privilege IAM roles

```
export const s3ReadWritePolicyForBucket = (bucketName: string) => new iam.PolicyStatement({
  actions: ['s3:PutObject', 's3>ListBucket', 's3:GetObject', 's3>DeleteObject'],
  resources: [`arn:aws:s3:::${bucketName}` , `arn:aws:s3:::${bucketName}/*`],
});

export const dynamodbGetPolicy = (tableName: string) => new iam.PolicyStatement({
  actions: ['dynamodb:GetItem', 'dynamodb:Scan'],
  resources: [`arn:aws:dynamodb:eu-central-1:*:table/${tableName}`]
});

export const sendEmailPolicy = (fromAddress: string) => new iam.PolicyStatement({
  actions: ['ses:SendEmail'],
  resources: ['*'],
  conditions: { 'StringEquals': { 'ses:FromAddress': fromAddress } }
});
```

Spot two errors

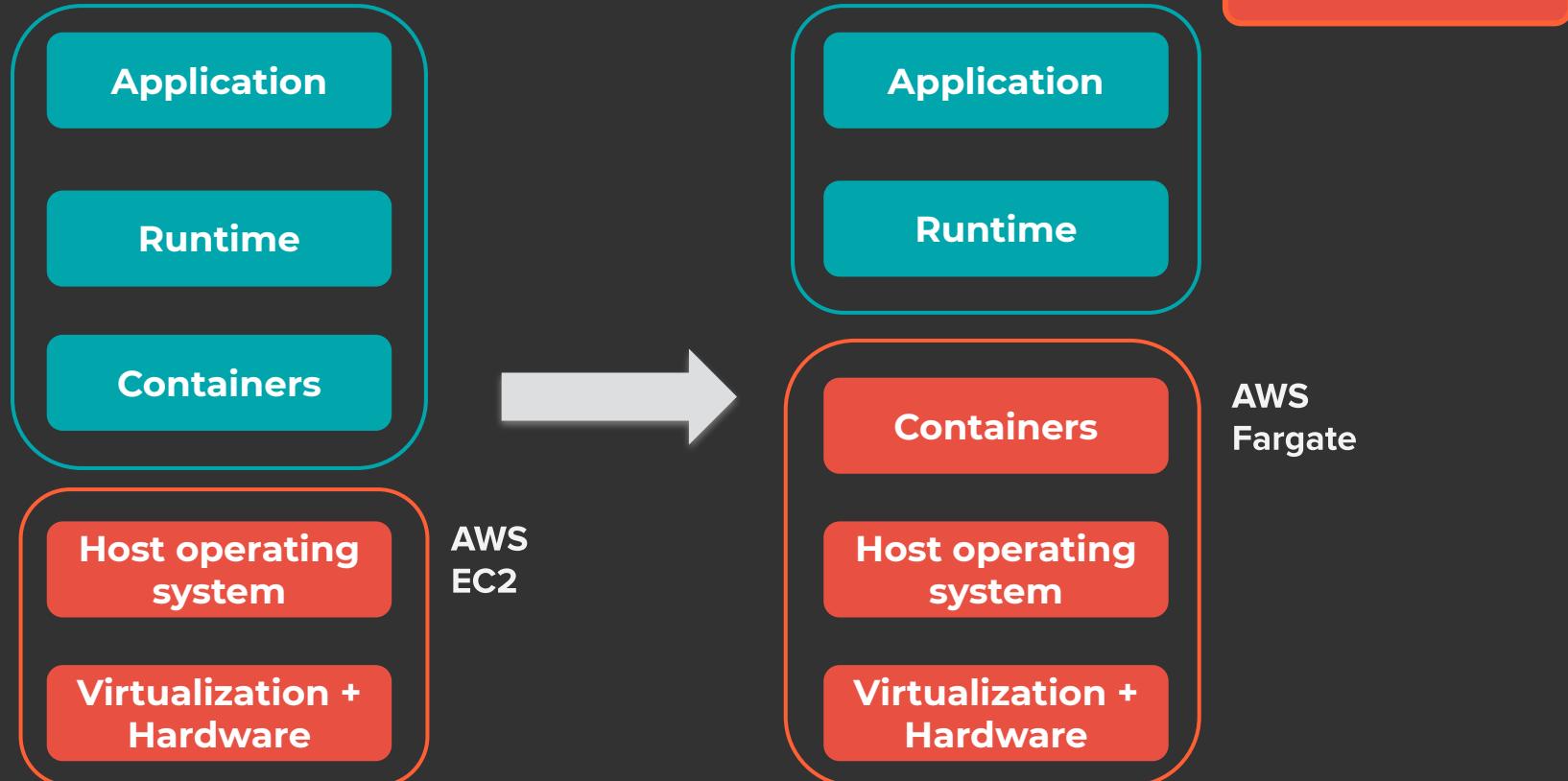
Sometimes errors unfortunately appear only at deployment time. Developers would benefit from compile-time validation whenever possible.

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

Automated container management



From “infrastructure as a service” to “containers as a service”



The **old way** of deploying clusters with never ending CloudFormation templates

Resources:

```

MasterSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: !Join ["-", ["master-sg", !Ref InfrastructureName]]
    GroupDescription: Cluster Master Security Group
    SecurityGroupIngress:
      - IpProtocol: icmp
        FromPort: 0
        ToPort: 0
        CidrIp: !Ref VpcCidr
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: !Ref VpcCidr
      - IpProtocol: tcp
        FromPort: 6443
        ToPort: 6443
        CidrIp: !Ref VpcCidr
      - IpProtocol: tcp
        FromPort: 22623
        ToPort: 22623
        CidrIp: !Ref VpcCidr
    VpcId: !Ref VpcId
  
```

WorkerSecurityGroup:

```

  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: !Join ["-", ["worker-sg", !Ref InfrastructureName]]
    GroupDescription: Cluster Worker Security Group
    SecurityGroupIngress:
      - IpProtocol: icmp
        FromPort: 0
        ToPort: 0
        CidrIp: !Ref VpcCidr
      - IpProtocol: tcp
  
```

```

MasterInstanceProfile:
  Type: "AWS::IAM::InstanceProfile"
  Properties:
    Path: "/"
    Roles:
      - Ref: "MasterIamRole"

WorkerIamRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: !Join ["-", [>Ref InfrastructureName, "worker", "role"]]
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: "Allow"
          Principal:
            Service:
              - "ec2.amazonaws.com"
          Action:
            - "sts:AssumeRole"
    Path: "/"
    Policies:
      - PolicyName: !Join ["-", [>Ref InfrastructureName, "worker", "policy"]]
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: "Allow"
              Action: "ec2:Describe*"
              Resource: "*"
  
```

Source: <https://github.com/Jackrokdj1/openshift/> (Apache 2.0)

The new way of setting up clusters

Using simplified AWS CDK constructs you get same functionality and security with less code.

Services are specified in terms of **CPU** and **memory** that are needed for each container.

Cluster management happens automatically by Fargate.

```
const vpc = new ec2.Vpc(this, 'ExampleVpc', { maxAzs: 2 });

const cluster = new ecs.Cluster(this, 'ExampleCluster', { vpc });

// Import existing image and route traffic with a public Network Load Balancer
new ecs_patterns.ApplicationLoadBalancedFargateService(this,
  "ExampleService", {
    cluster,
    cpu: 512,
    memoryLimitMiB: 2048,
    desiredCount: 2,
    image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
    publicLoadBalancer: true
});
```

AWS CDK is already becoming a norm



Richard H. Boyd
@rchrdbdyd

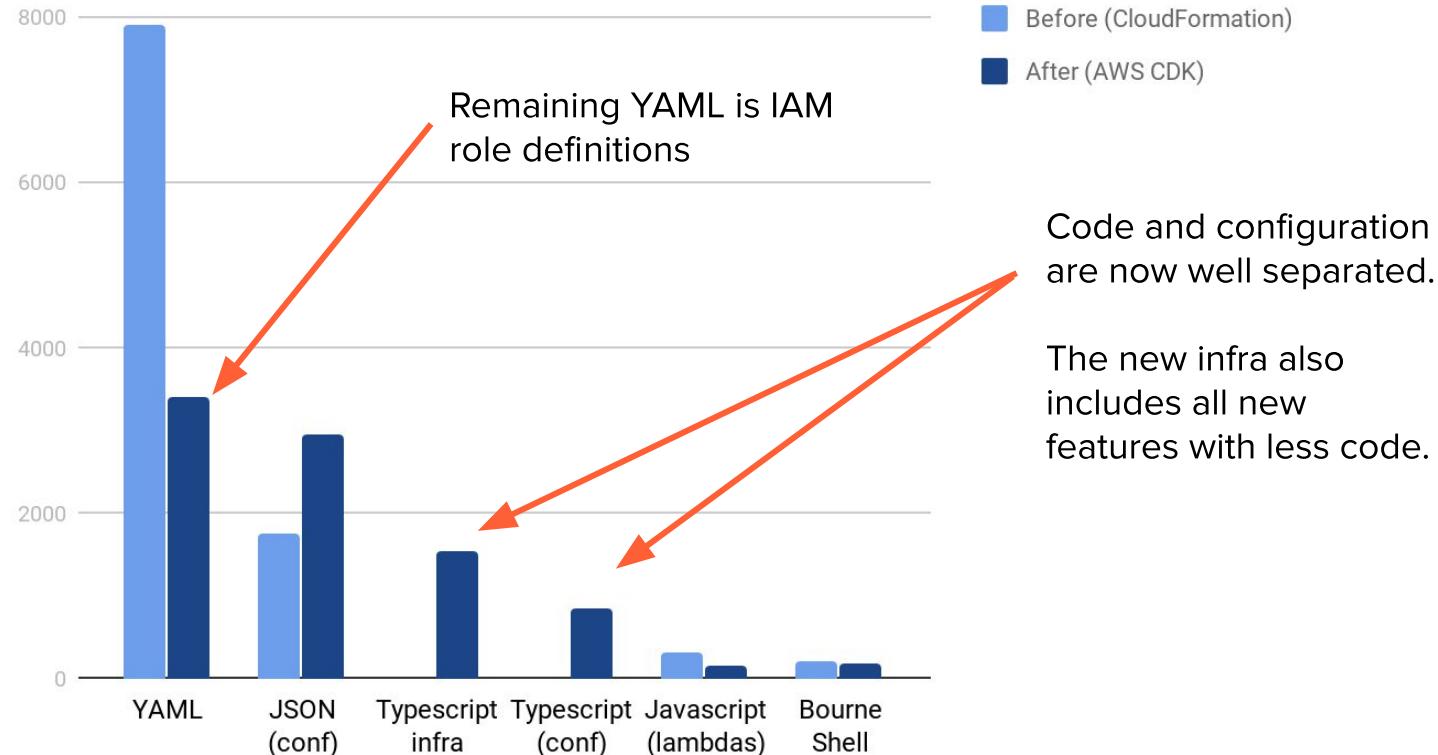
I saw a guy building his infrastructure today.
No type-safety.
No resource validation.
No higher level abstractions.
No CDK
He just sat there crying.
Writing cloudformation in a text-editor.
Like a Psychopath.

11:13 PM · Oct 25, 2019 · Twitter Web App

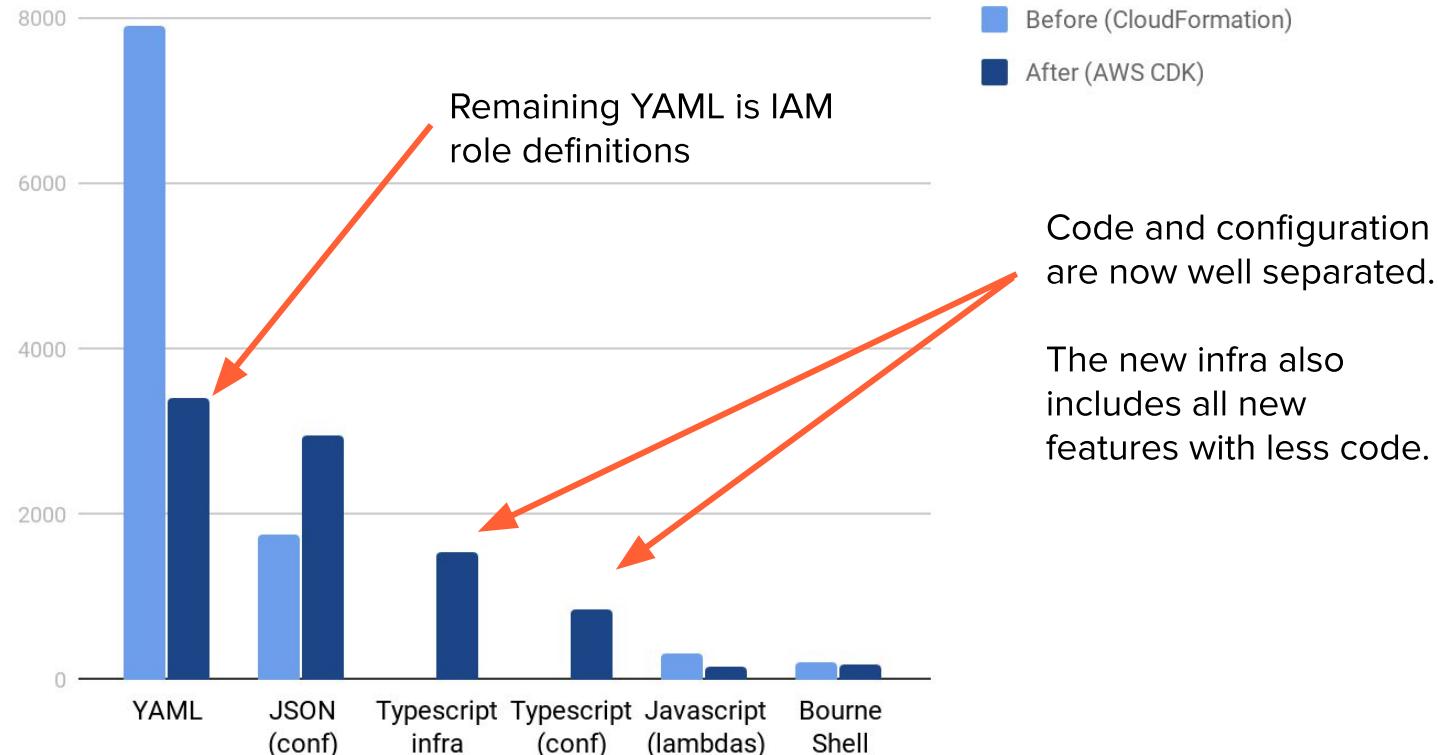
<https://twitter.com/rchrdbdyd/status/1187824437544050688>

Conclusions

Infrastructure code before and after migration



Infrastructure code before and after migration



Executive summary

Experiences from AWS CDK migration in a hybrid architecture.

1. **Cost savings** by using industry best practices
2. **Cloud First:** Outsource container and database management
3. **Faster** development cycle from development to production
4. Increased **information security** and **compliance**
5. Using managed services: **Pay for only what you use**
6. **43% less code** to maintain!



Discussion

Q&A



@markuslindqv



Thank you.

markus.lindqvist@reaktor.com