DEV328

DevOps Lessons from **Coursera**: Site Performance, Reliability, and Developer Productivity

Lewis Chung + Bryan Kane

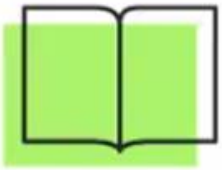
Frontend Infrastructure @ Coursera

November 28, 2017

We envision a world where anyone, anywhere can transform their life by accessing the world's best learning experience.

30+ million
learners

150+
partners

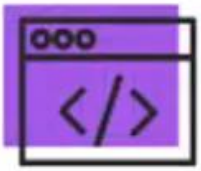


2,200+
courses

AWS CodeBuild
Optimizing application builds

Amazon ECS
Isolating front-end applications

Application



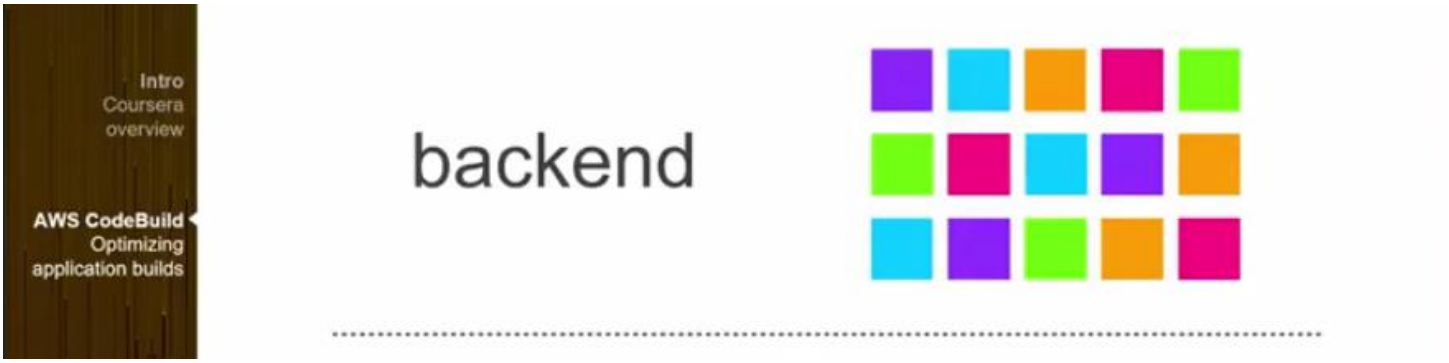
70+
engineers

Keep Coursera as **fast** and **reliable** as possible.

Empower Coursera's engineers to be as **productive** as possible.

Adding build flexibility and reducing costs with

AWS CodeBuild



Backend has always been developed as microservices

But the frontend was still one big JS monolith

Coursera in 2013

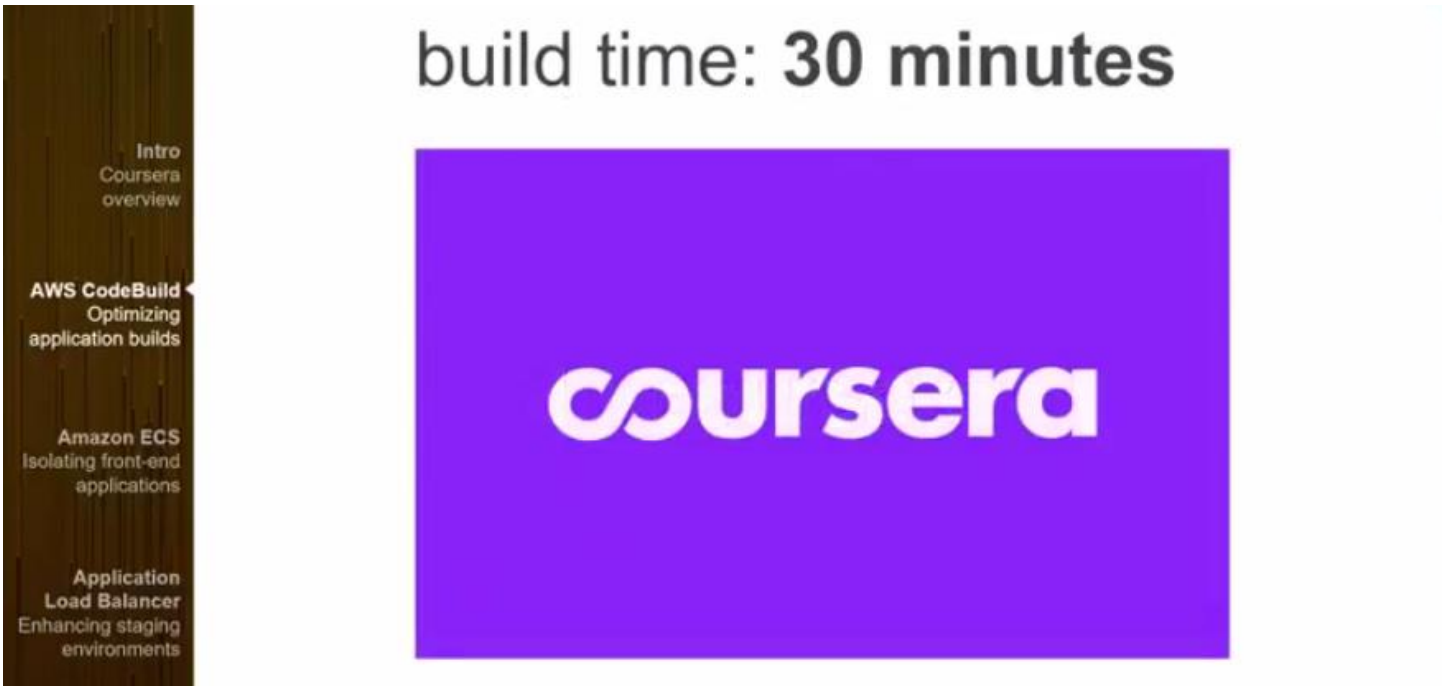
- commit to deploy in less than 5 minutes
- commit and deploy happen together
- regression debugging is easy w/ granular deploys

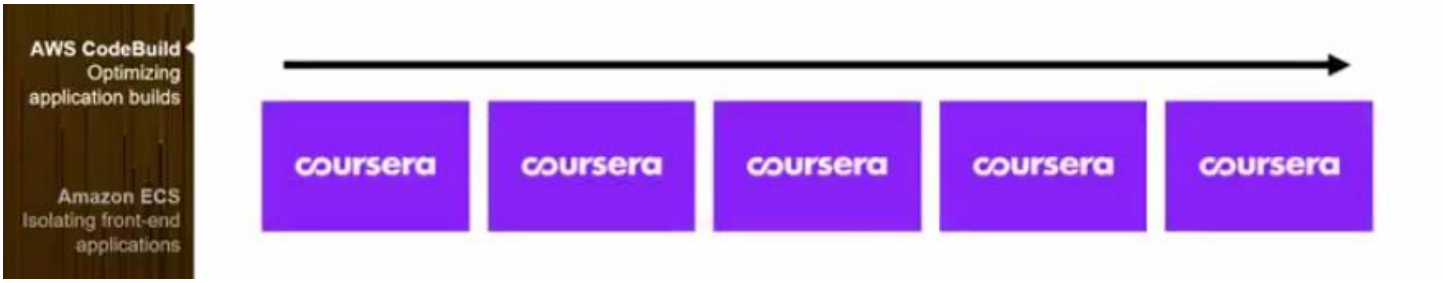
build time: **5 minutes**

coursera

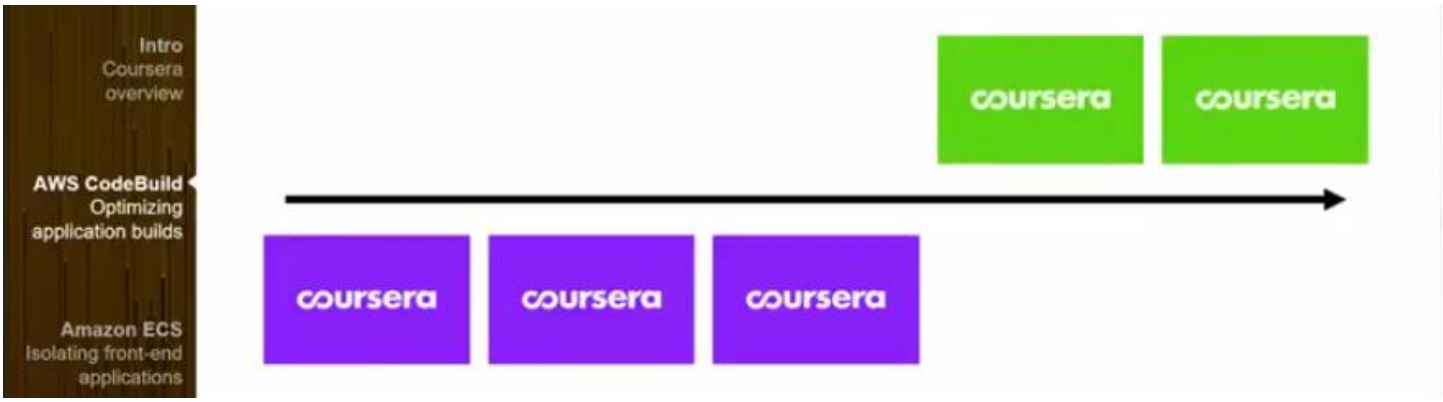
build time: **15 minutes**

coursera



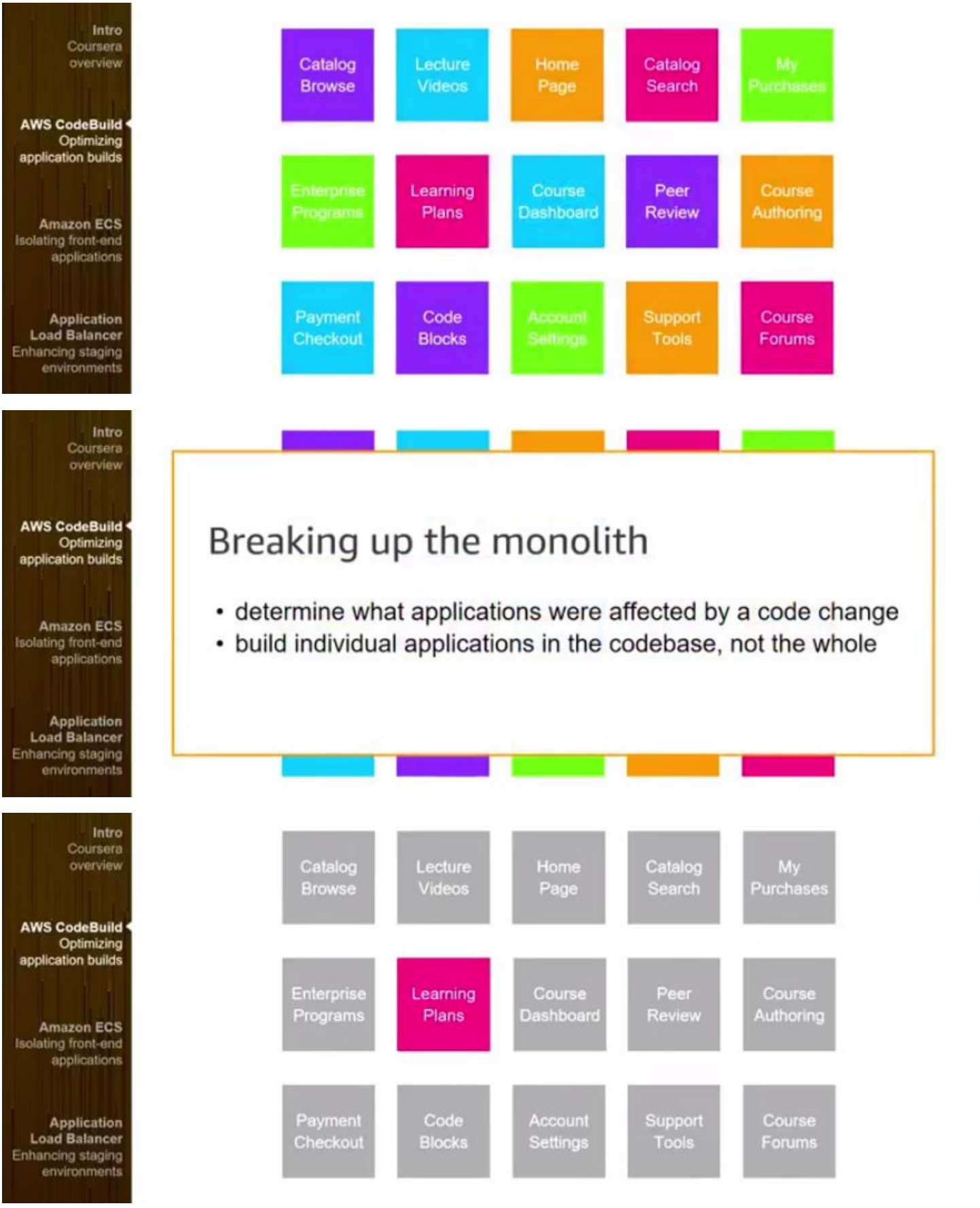


Say we have 5 feature commits at once that are ready for testing and deployment



If 3rd commit has a problem

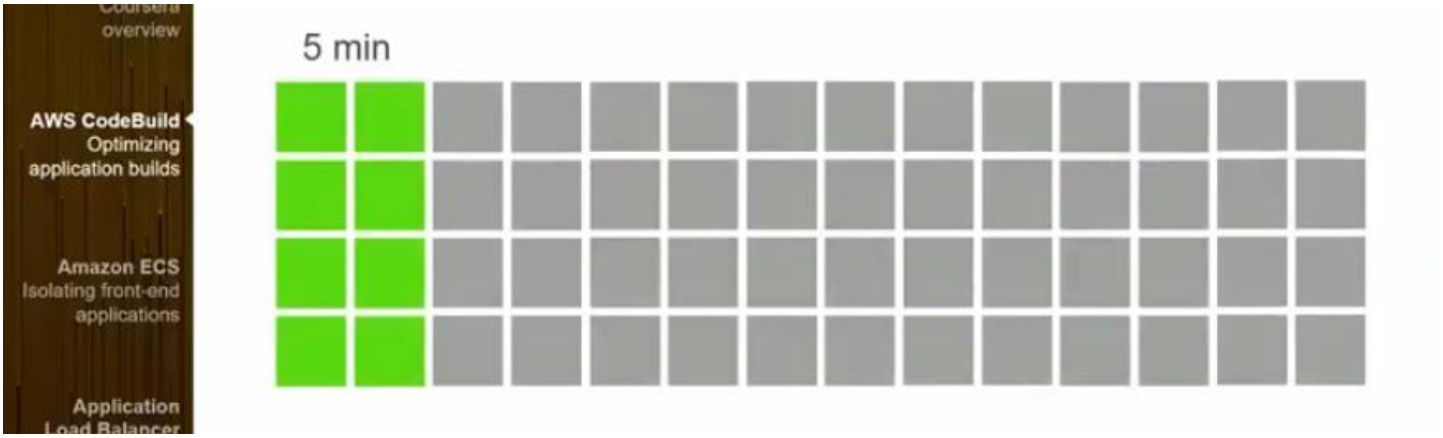
We can't deploy them all, we need to break them out and deploy one by one. So, we decided to break our frontend monolith up


Intro
Coursera overview
AWS CodeBuild
Optimizing application builds
Amazon ECS
Isolating front-end applications
Application Load Balancer
Enhancing staging environments
Catalog Browse
Lecture Videos
Home Page
Catalog Search
My Purchases
Enterprise Programs
Learning Plans
Course Dashboard
Peer Review
Course Authoring
Payment Checkout
Code Blocks
Account Settings
Support Tools
Course Forums
Breaking up the monolith
• determine what applications were affected by a code change
• build individual applications in the codebase, not the whole

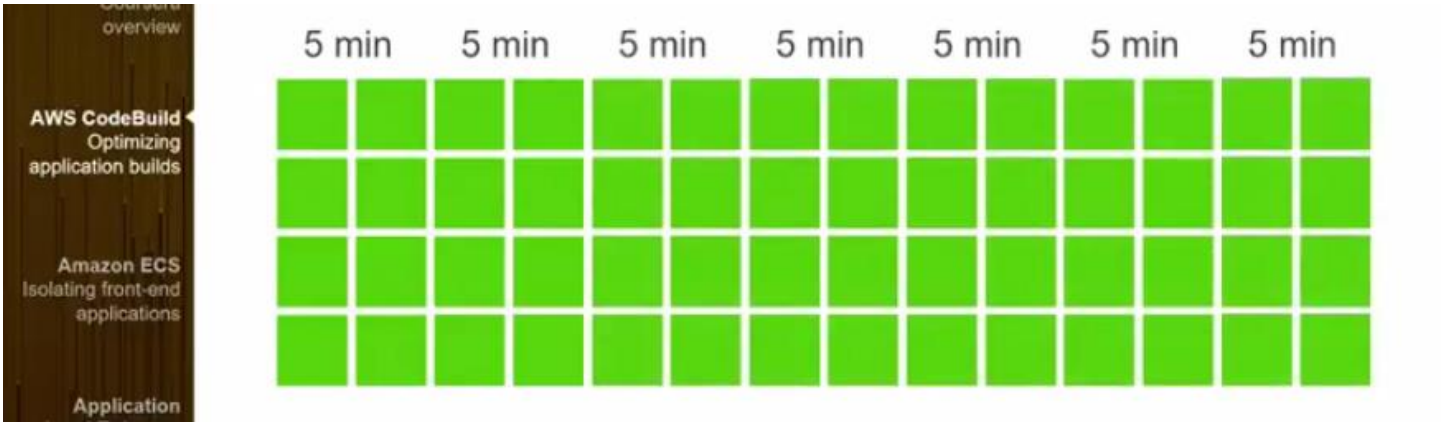
We now build mSes that are affected by a feature commit

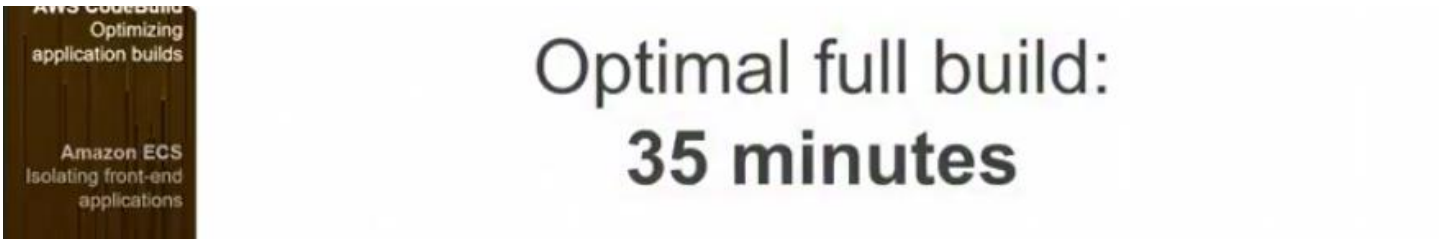
Except for features that affect all mSes like the page header

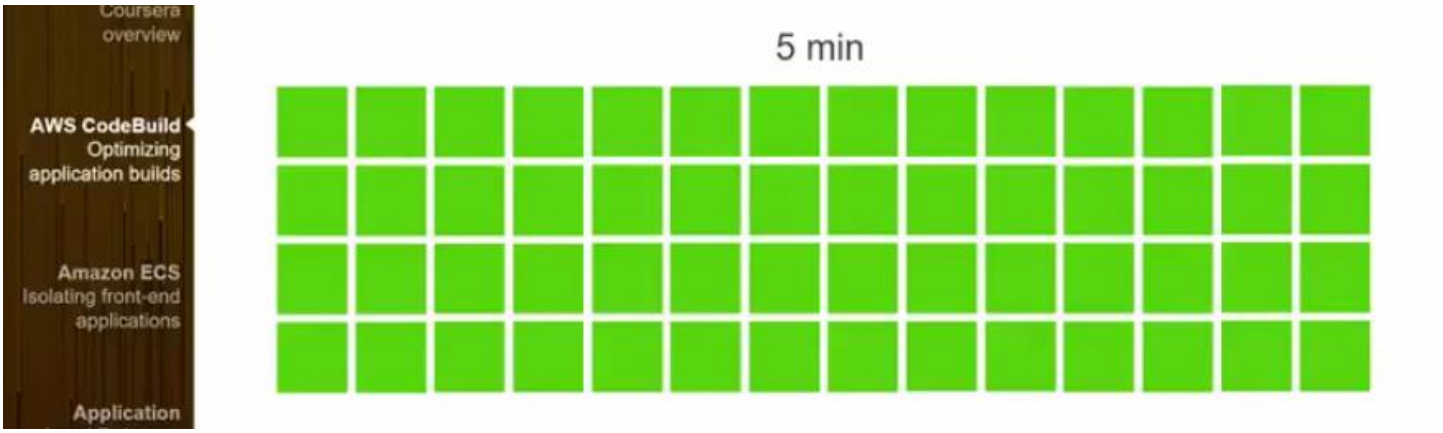
We kept a pool of 8 workers on EC2 that monitors our build queue



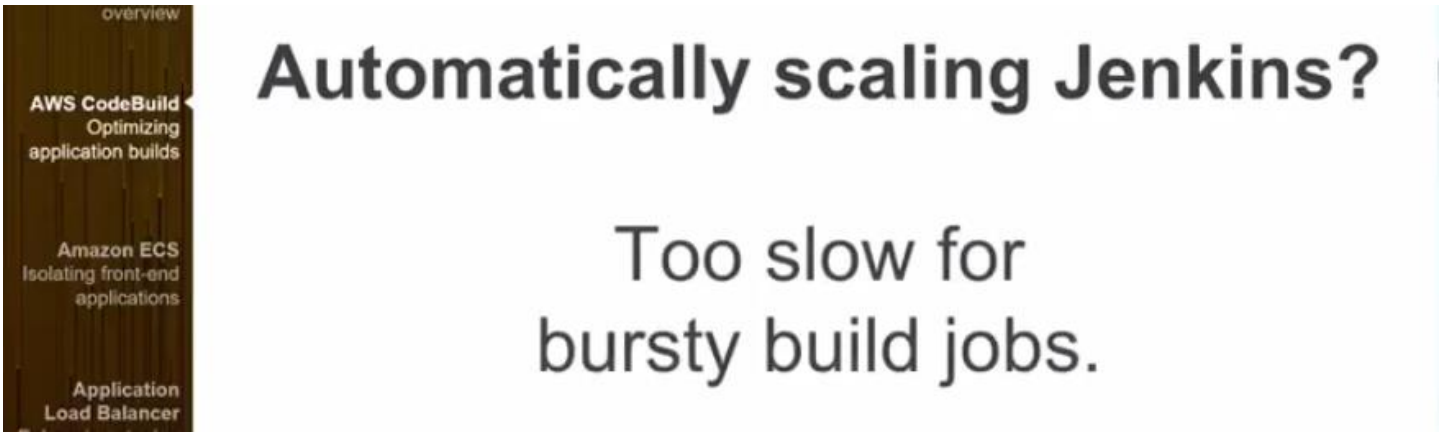
When we have 50 builds in the build queue



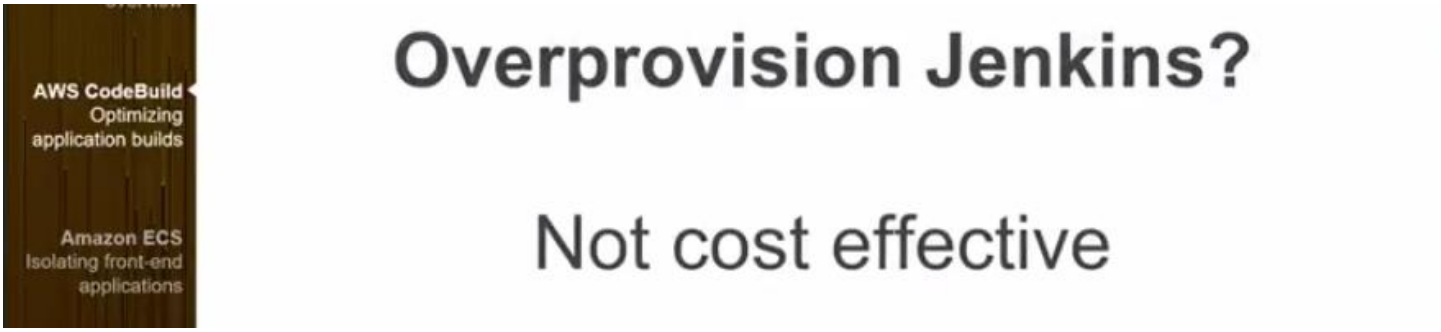


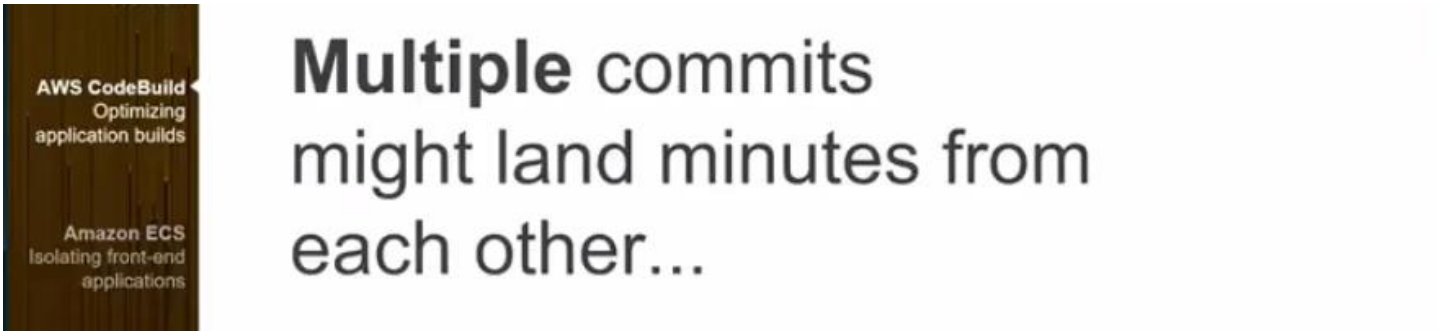


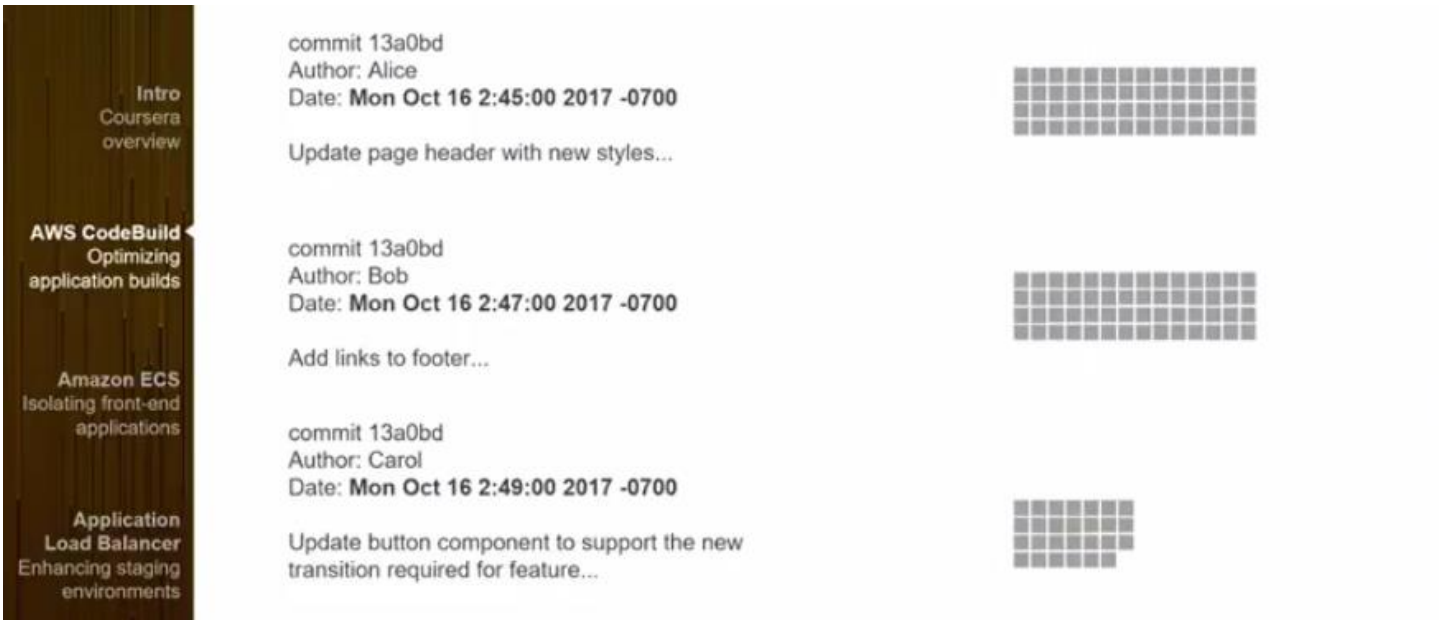
This is what we actually want, building everything in about 5 minutes



Can we automatically scale our Jenkins worker pool quickly enough to do this?







This scenario shows that we have over 100 builds queued up in the build system and 8 workers is not enough for 5 mins

Can we **do better?**

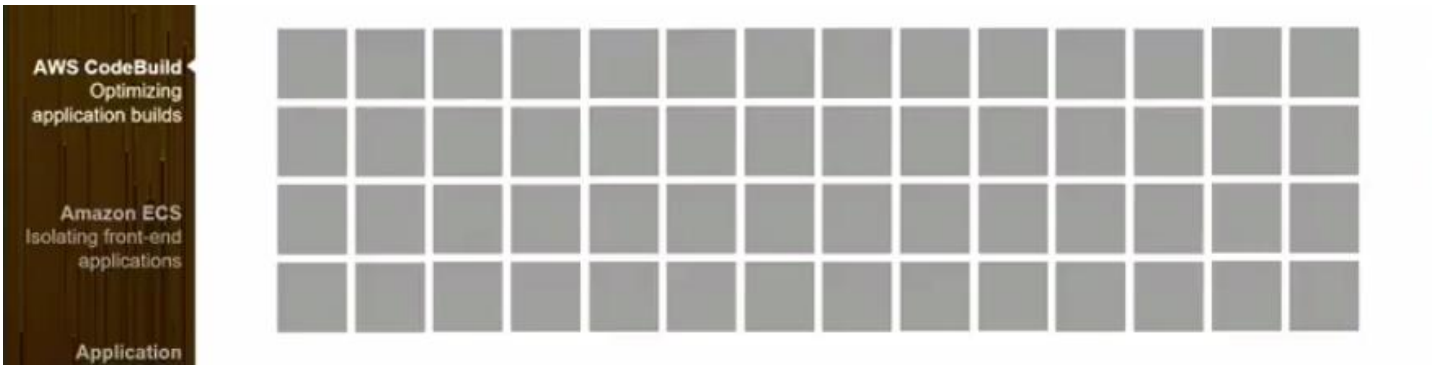
AWS CodeBuild

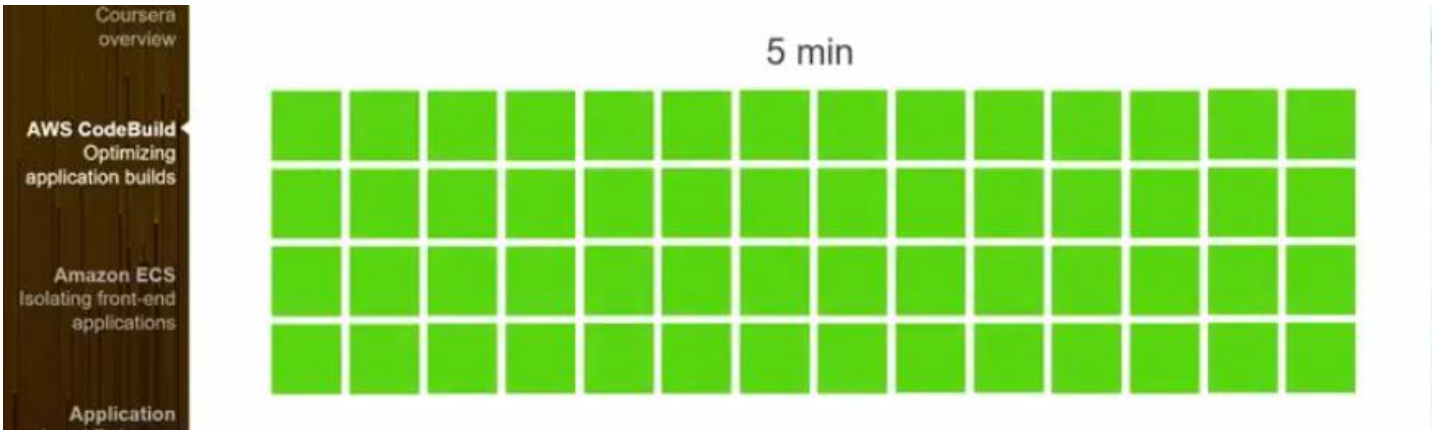
AWS CodeBuild

What is **CodeBuild?**

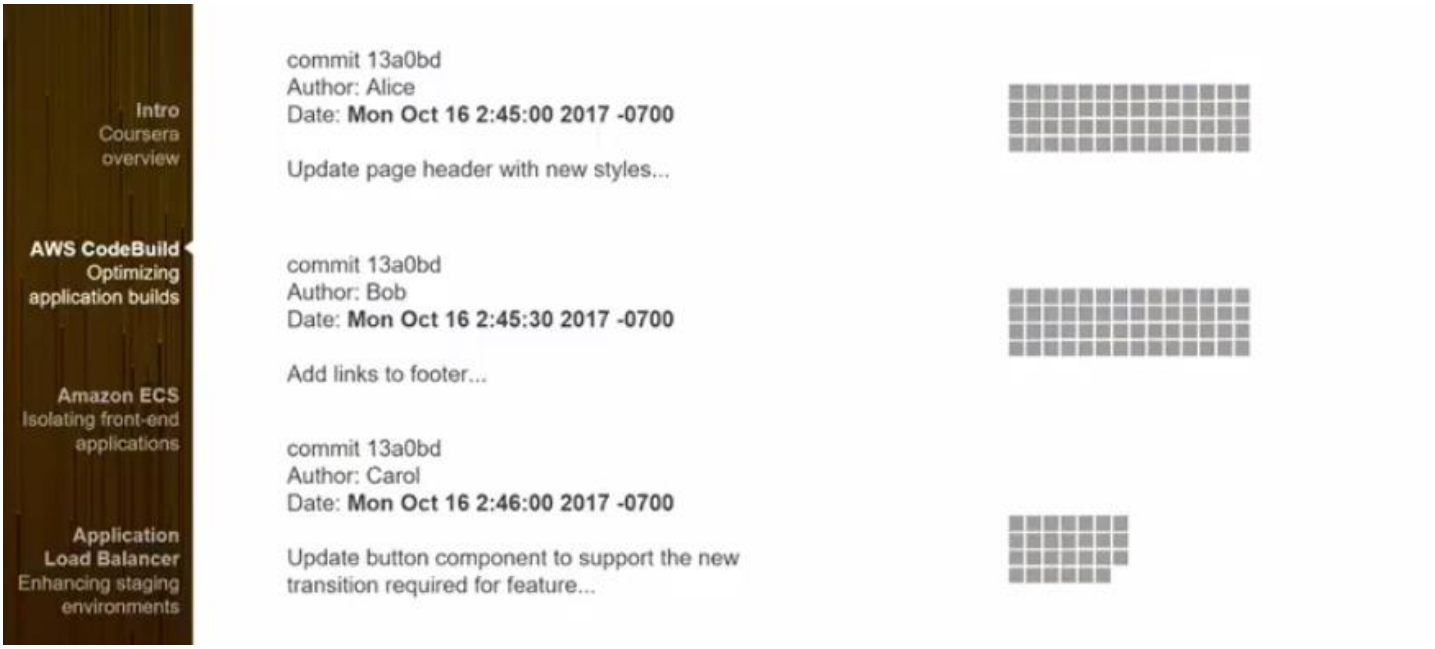
- on-demand build environment
- run your builds inside docker containers
- near-infinite elastic capacity -- scale up and down as you need

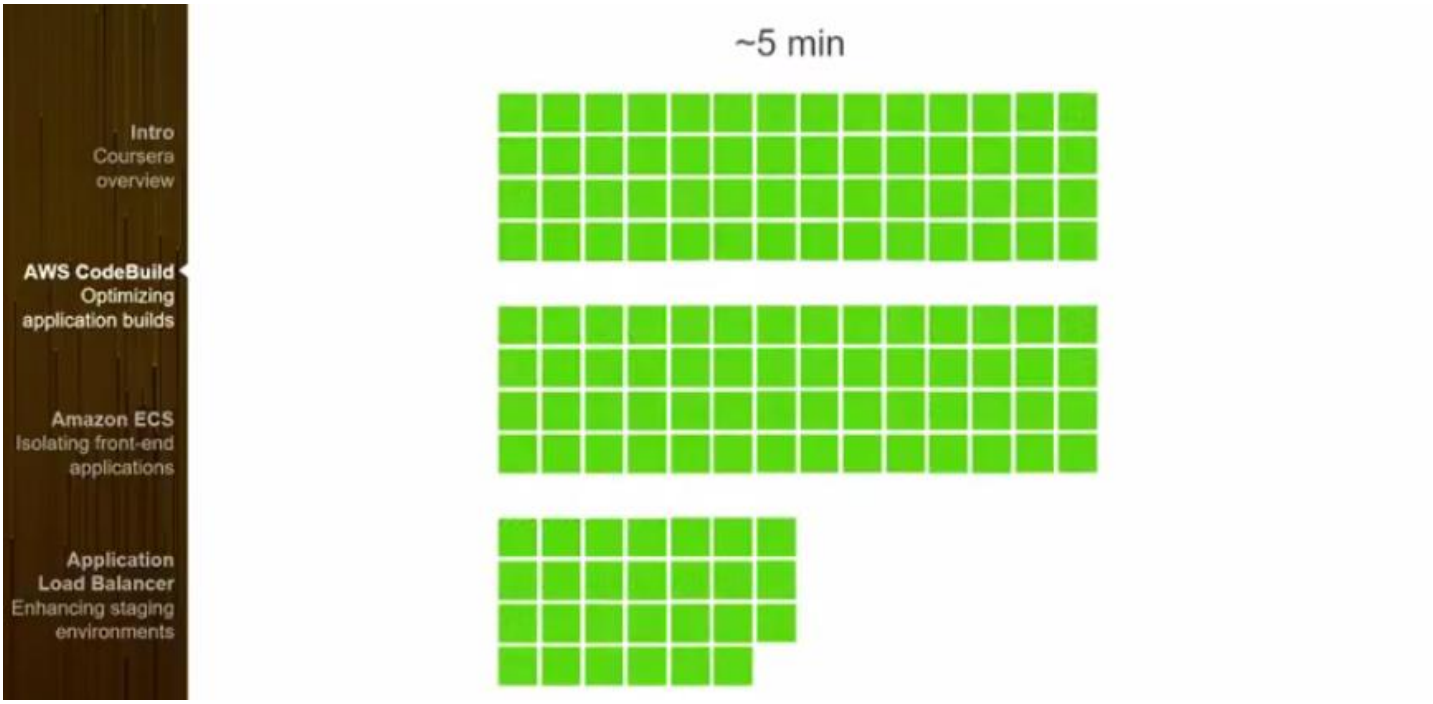
Compute instance type	Memory (GB)	vCPU	Price per build minute ($)
small	3	2	0.005
medium	7	4	0.010
large	15	8	0.020





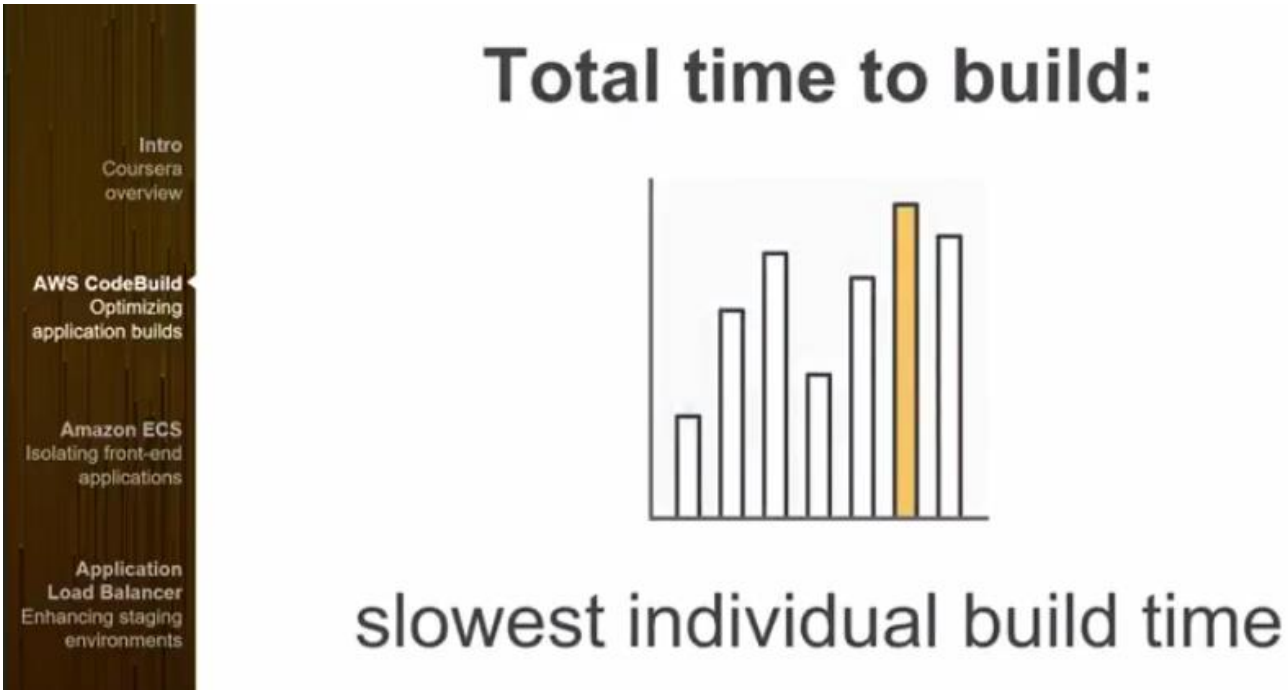
Now we can go from nothing being built to everything being built in 5 min…we can!





We can now get this done too

We were able to take our build scripts that we are running on Jenkins and make a few minor modifications to get it running on CodeBuild to make it compliant. Check the documentation



Intro
Coursera

Tips and Tricks

Intro
Coursera
overview

AWS CodeBuild
Optimizing
application builds

Amazon ECS
Isolating front-end
applications

Application
Load Balancer
Enhancing staging
environments

Build Visualization

Intro
Coursera
overview

AWS CodeBuild
Optimizing
application builds

Amazon ECS
Isolating front-end
applications

Application
Load Balancer
Enhancing staging
environments

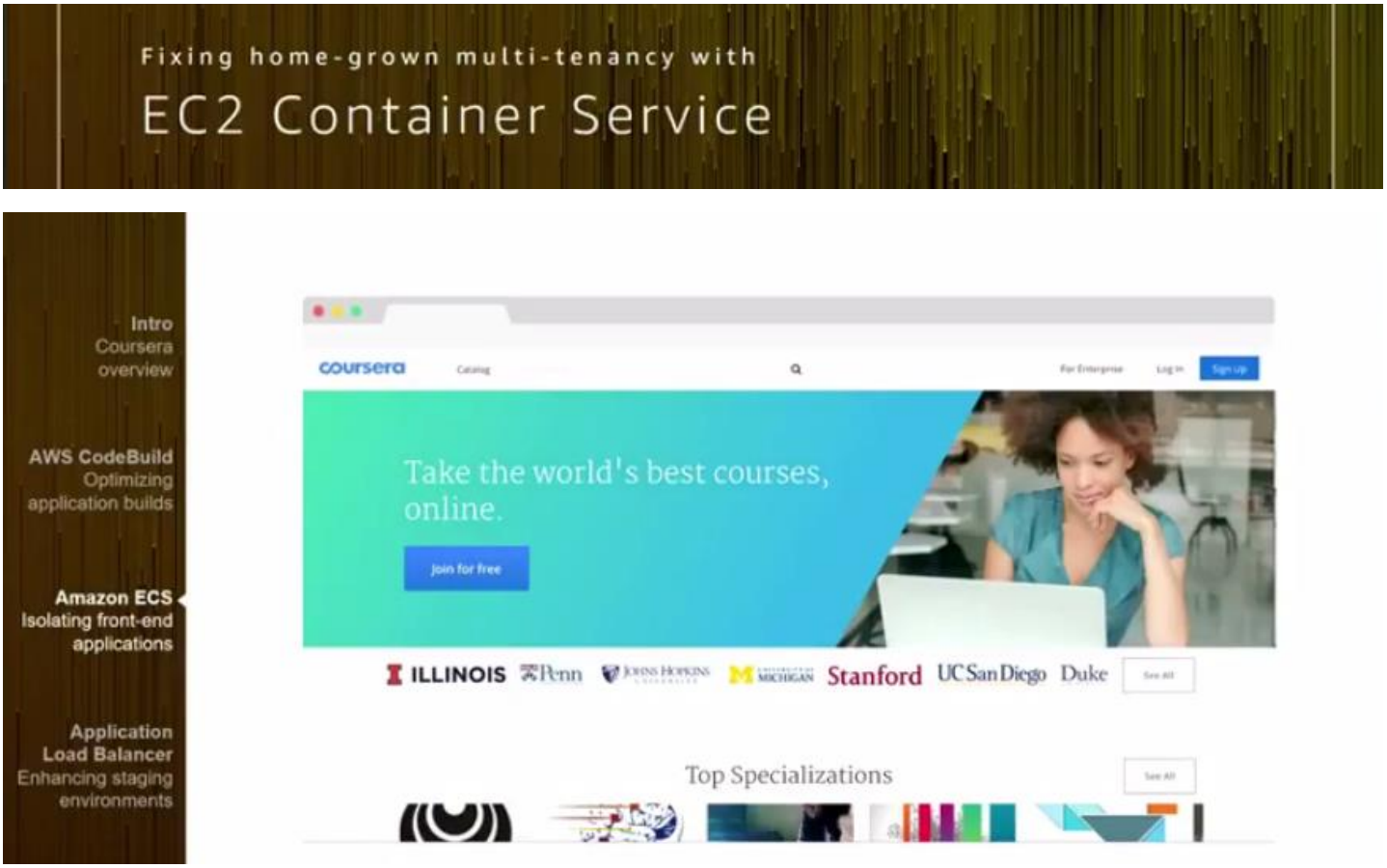
Supercharge Logs

```
35  09/27/2017 20:36:43.000-0700  [36m> Verifying integrity...[0m
36  09/27/2017 20:36:43.000-0700  gpg: keyring `/root/.gnupg/secring.gpg' created
37  09/27/2017 20:36:43.000-0700  gpg: key 86E50310: public key "Yarn Packaging <yarn@dan.cx>" imported
38  09/27/2017 20:36:43.000-0700  gpg: Total number processed: 1
39  09/27/2017 20:36:43.000-0700  gpg:               imported: 1  (RSA: 1)
40  09/27/2017 20:36:43.000-0700  gpg: Signature made Mon Feb 27 15:37:57 2017 UTC using RSA key ID FD2497F5
41  09/27/2017 20:36:43.000-0700  gpg: Good signature from "Yarn Packaging <yarn@dan.cx>"
42  09/27/2017 20:36:43.000-0700  gpg: WARNING: This key is not certified with a trusted signature!
43  09/27/2017 20:36:43.000-0700  gpg:          There is no indication that the signature belongs to the
    owner.
44  09/27/2017 20:36:43.000-0700  Primary key fingerprint: 72EC F46A 56B4 AD39 C907  B8B7 1646 B01B 86E5 0310
45  09/27/2017 20:36:43.000-0700  Subkey fingerprint: 6A87 0CC1 6680 6599 AA17  F881 46C2 130D FD24 97F5
```

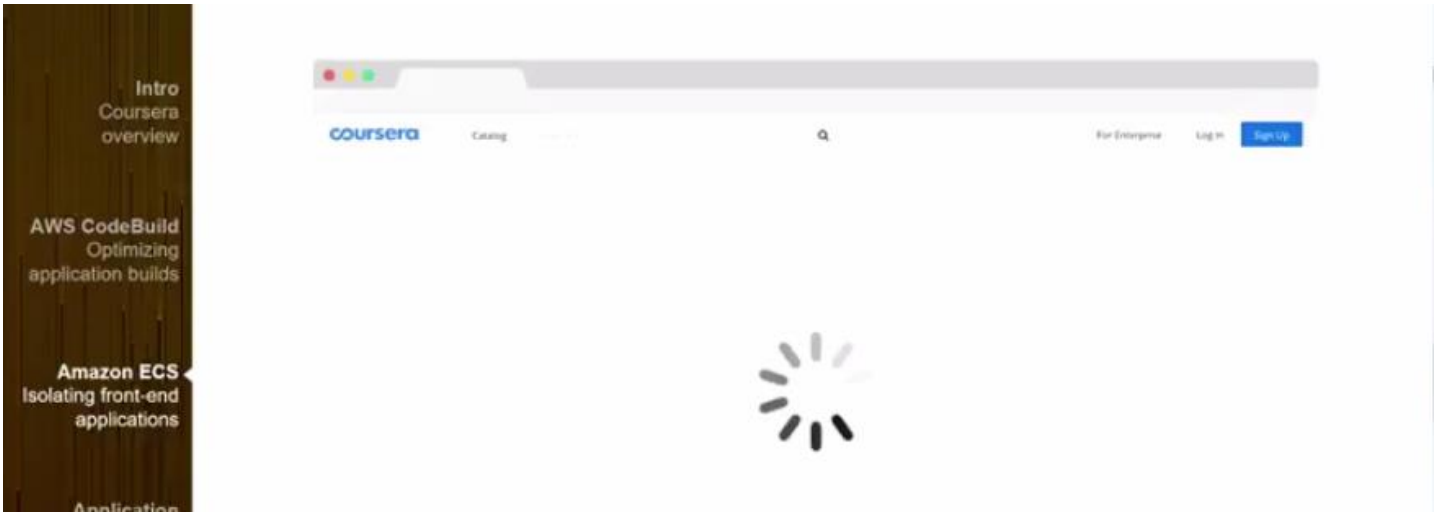
AWS CodeBuild
Optimizing
application builds

Amazon ECS
Isolating front-end

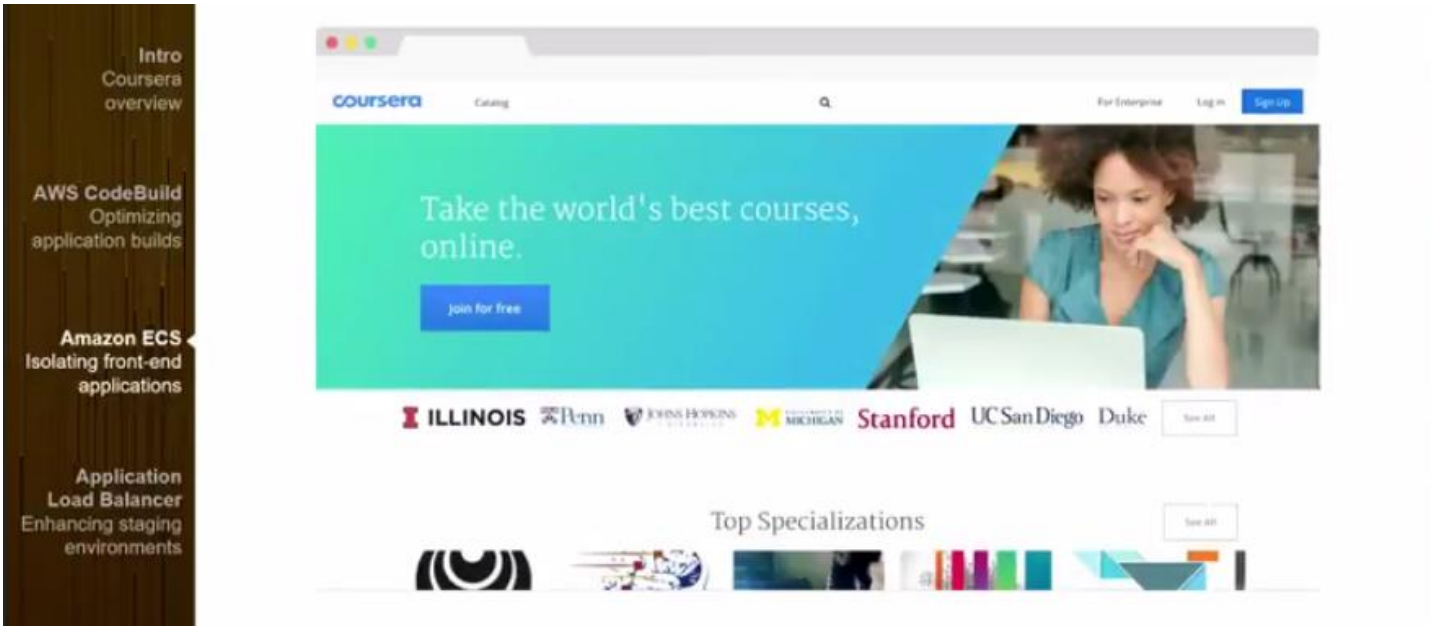
awslabs / **aws-codebuild-jenkins-plugin**



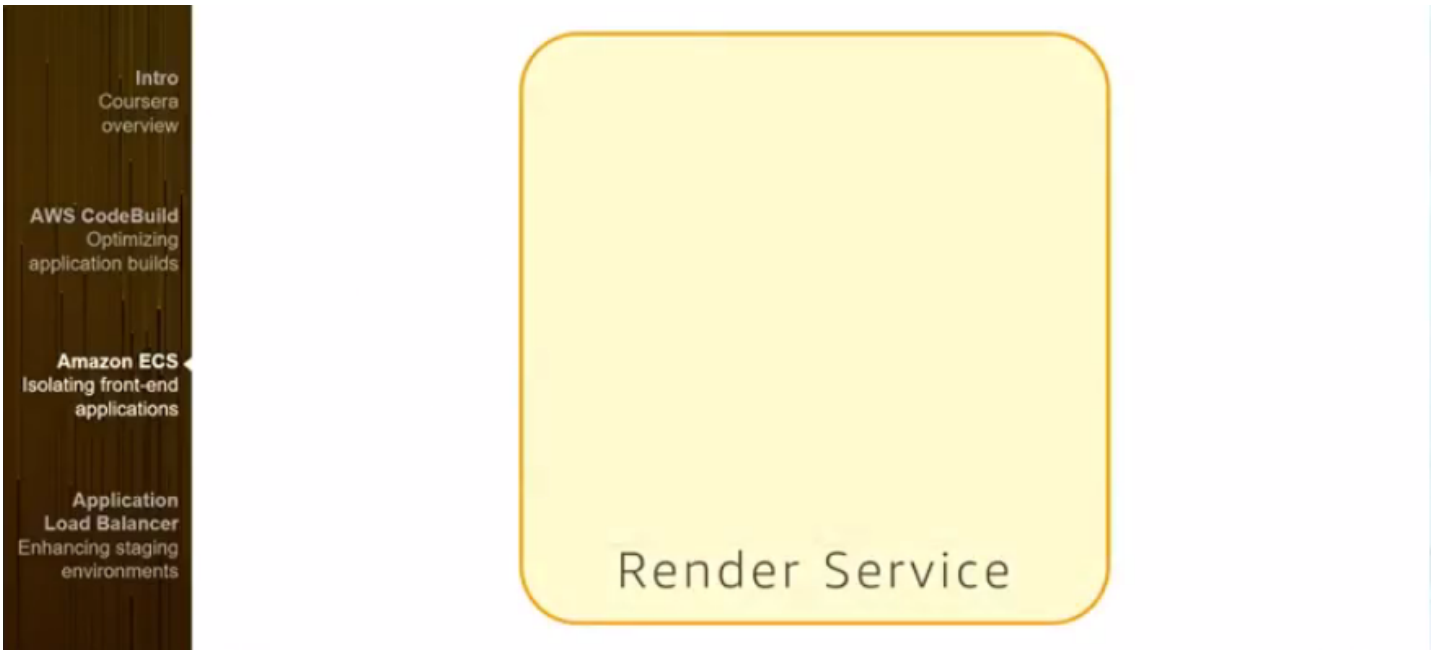
Coursera's website is an SPA architecture built with JS and React



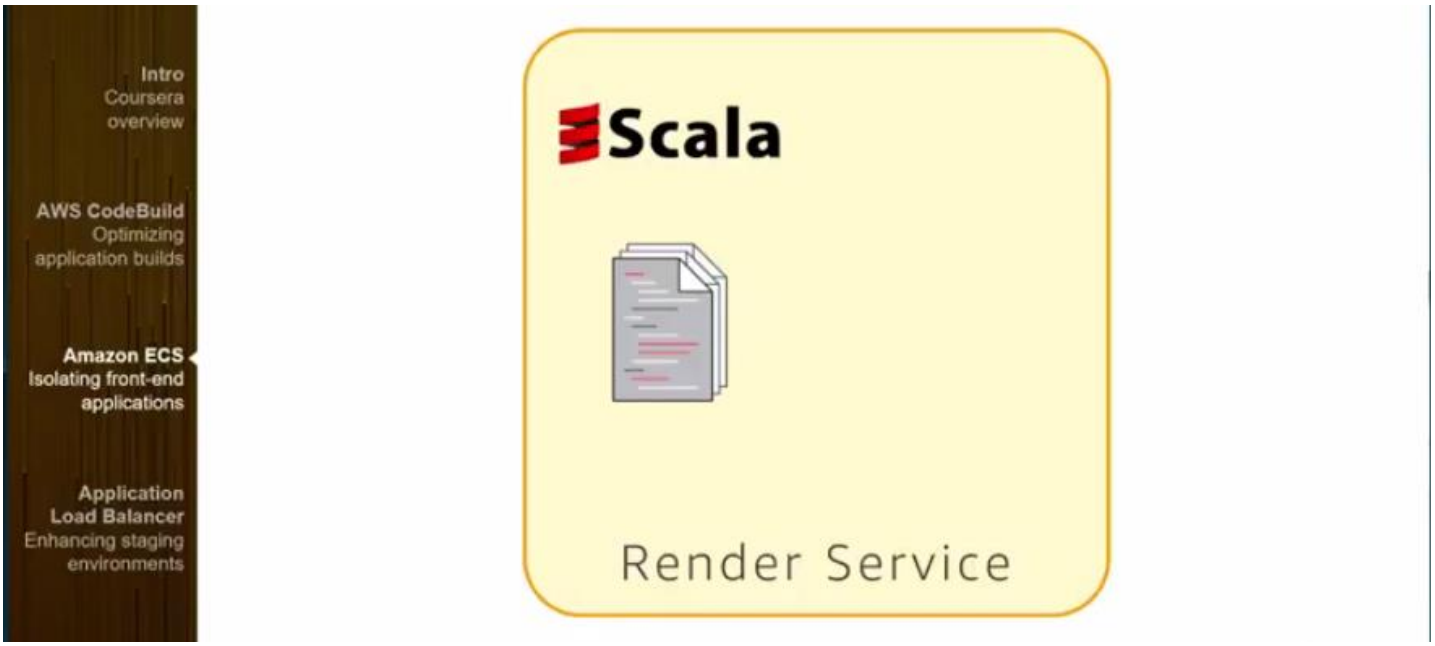
First you have JS downloaded, then it needs to make API calls for the content, then you get the full render



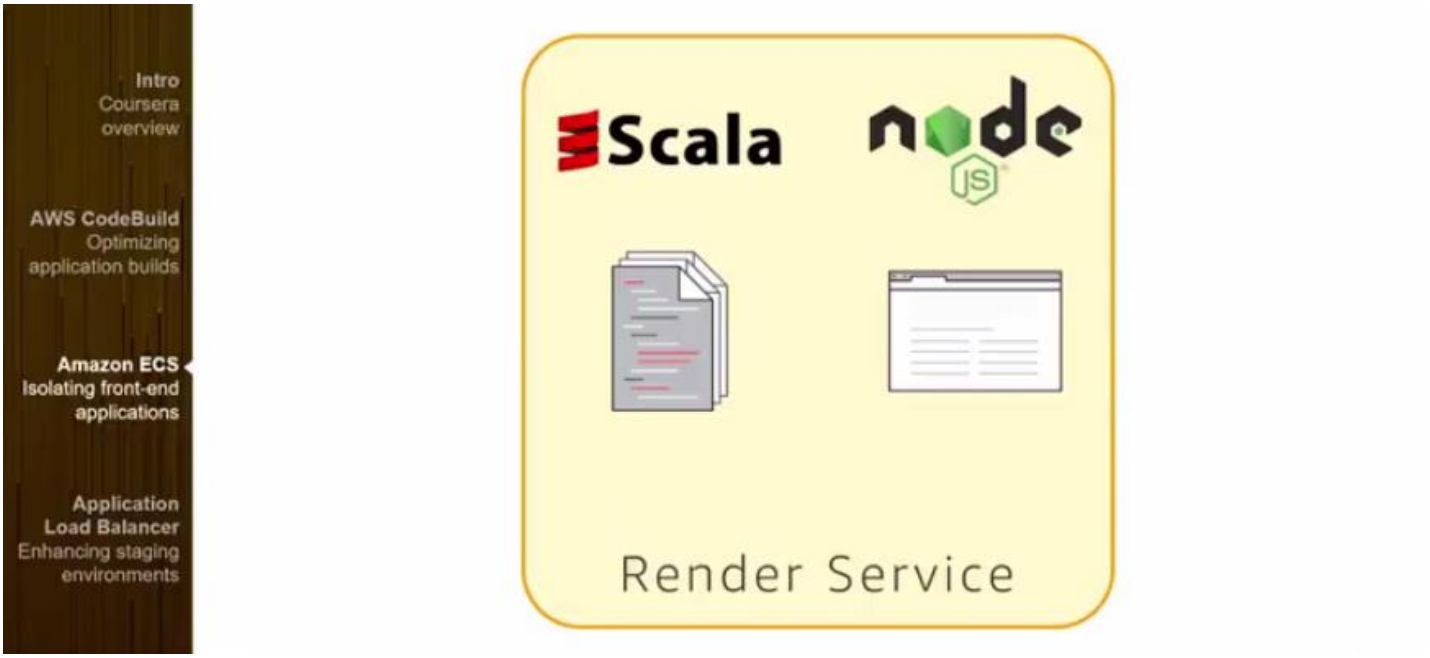
Using server-side rendering in React allows us to execute NodeJS on the server and make all the needed API calls on the server but then send fully rendered HTML content to the client. The new server-side approach is on the right side.



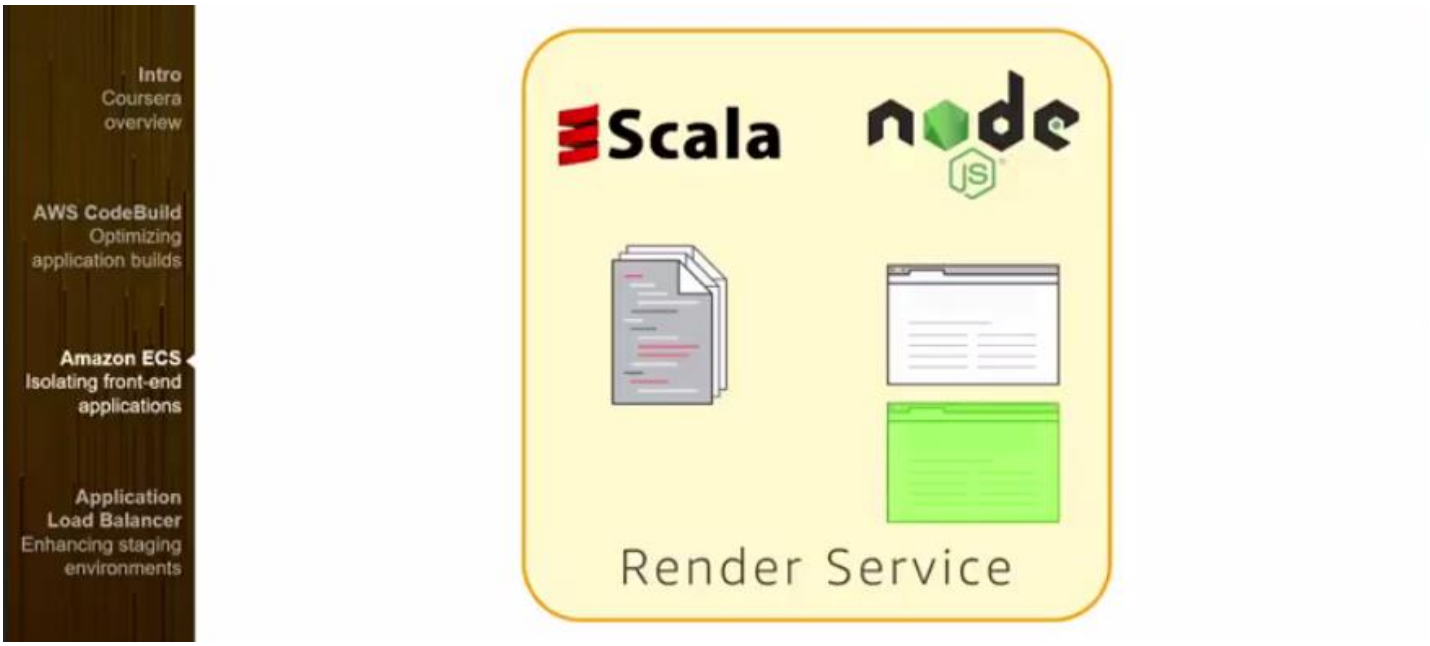
We now need to build a service to get this fully rendered pages



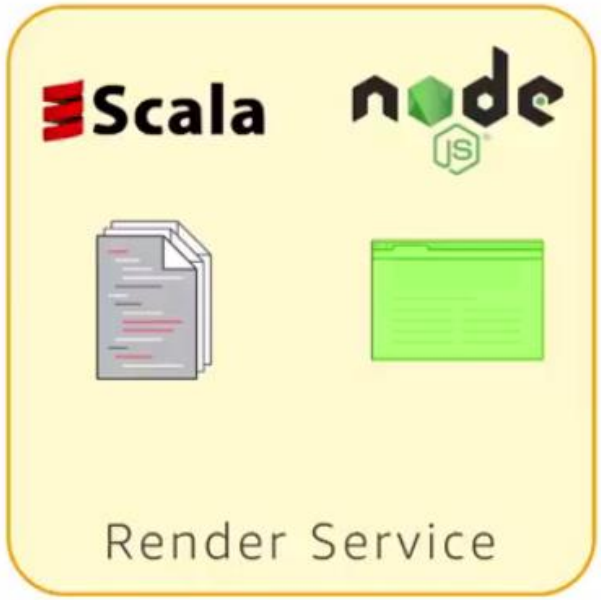
The Render service was built as a Scala service, which can’t really execute JS.

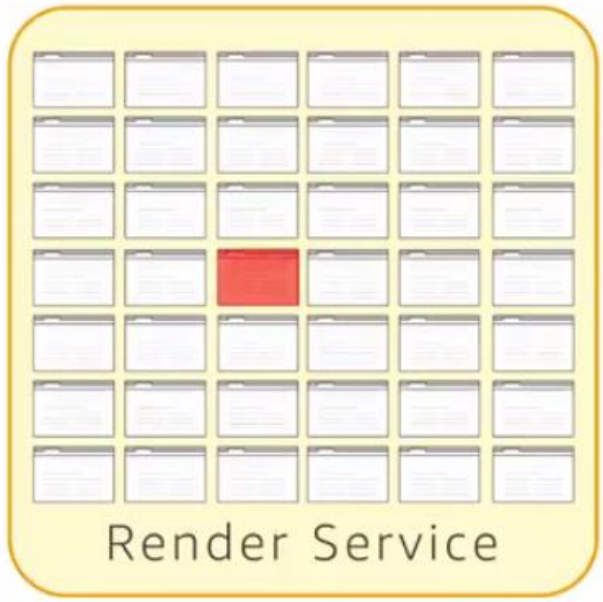


But NodeJS is good at executing JS, so we added some NodeJS to the Scala service. It is a NodeJS co-process to run alongside the Scala service with the NodeJS executing the JS and the Scala part doing everything else.

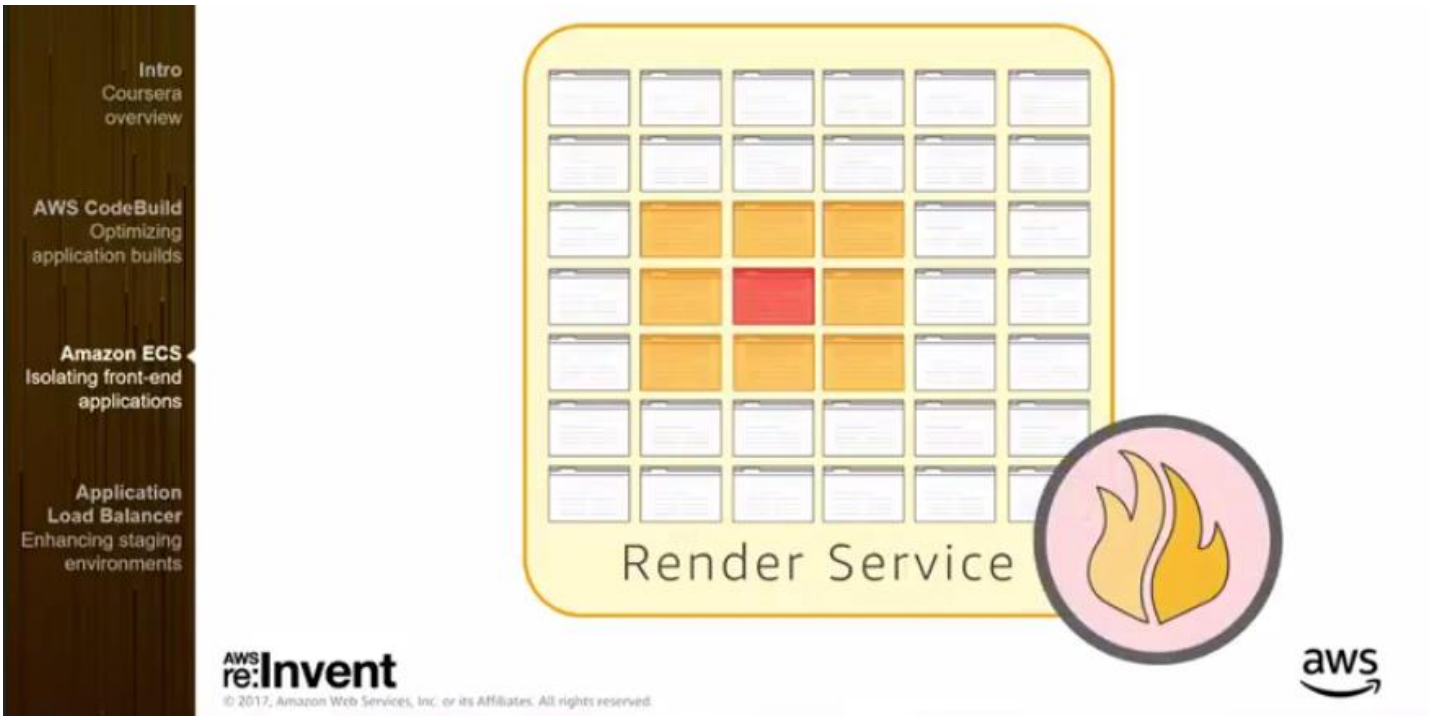


We do blue green deployment of the JS part before shifting traffic over to the new version



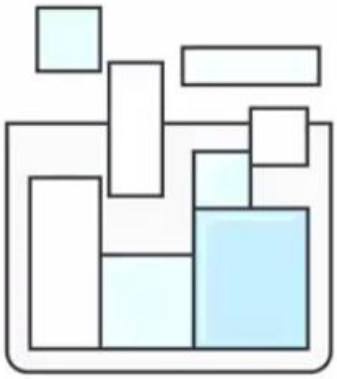


An application breaks occasionally



We want microservice isolation and not have faults bleed over to other microservice applications on the same EC2 host.

We can now run multiple containers on a single host

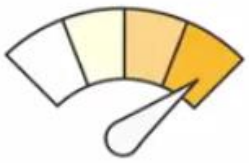


Multi-tenancy *and* isolation

AWS CodeBuild
Optimizing application builds

Amazon ECS
Isolating front-end applications

Application Load Balancer
Enhancing staging environments



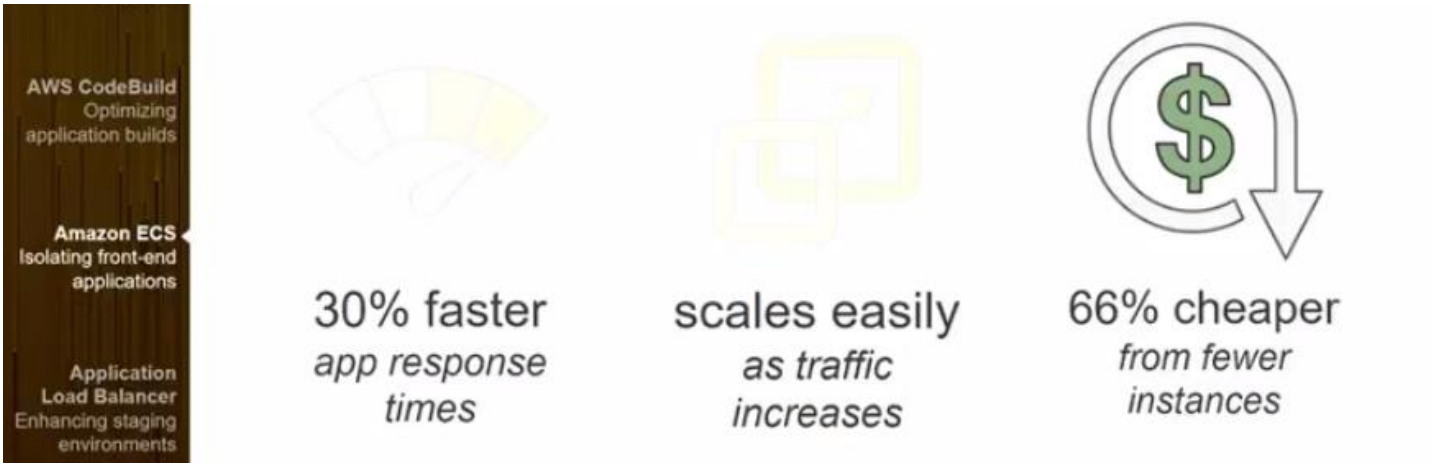
30% faster
app response times

scales easily
as traffic increases

66% cheaper
from fewer instances


AWS CodeBuild
Optimizing application builds
Amazon ECS
Isolating front-end applications
Application Load Balancer
Enhancing staging environments
30% faster
app response times
scales easily
as traffic increases
66% cheaper
from fewer instances


AWS CodeBuild
Optimizing application builds
Amazon ECS
Isolating front-end applications
Application Load Balancer
Enhancing staging environments
Oprah Winfrey
@Oprah
Follow
How to teach character! Fascinated by online course organized by my friend @norman_atkins @RelayGSE.
coursera.org/course/teachin…


AWS CodeBuild
Optimizing application builds
Amazon ECS
Isolating front-end applications
Application Load Balancer
Enhancing staging environments
30% faster
app response times
scales easily
as traffic increases
66% cheaper
from fewer instances

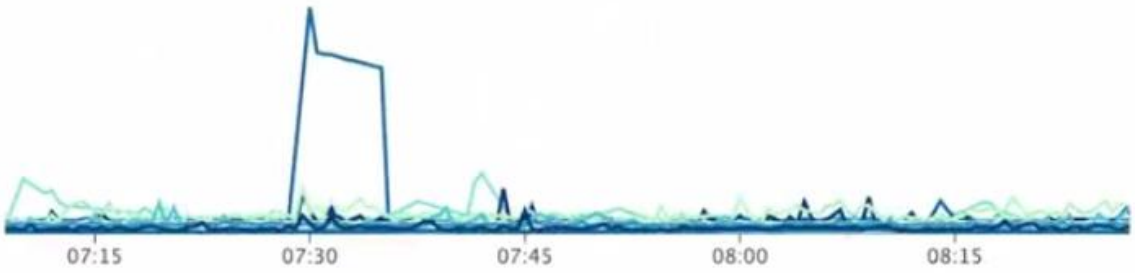
Intro
Coursera
overview

AWS CodeBuild
Optimizing
application builds

Amazon ECS
Isolating front-end
applications

Application
Load Balancer

Application isolation prevents cascading failures



AWS CodeBuild
Optimizing
application builds

Amazon ECS
Isolating front-end
applications

Application

one developer

migrated everything in

two months

Intro
Coursera
overview

AWS CodeBuild
Optimizing
application builds

Scale your containers *and* hosts

Choose one metric to scale your containers on:

pick either **CPU** or **memory**

Scale your containers *and* hosts

Combine the metrics for your ECS hosts:
scale based on *potential* number of
new containers that could be launched

http://garbe.io/blog/2017/04/12/
a-better-solution-to-ecs-autoscaling/

Intro
Coursera overview

AWS CodeBuild
Optimizing application builds

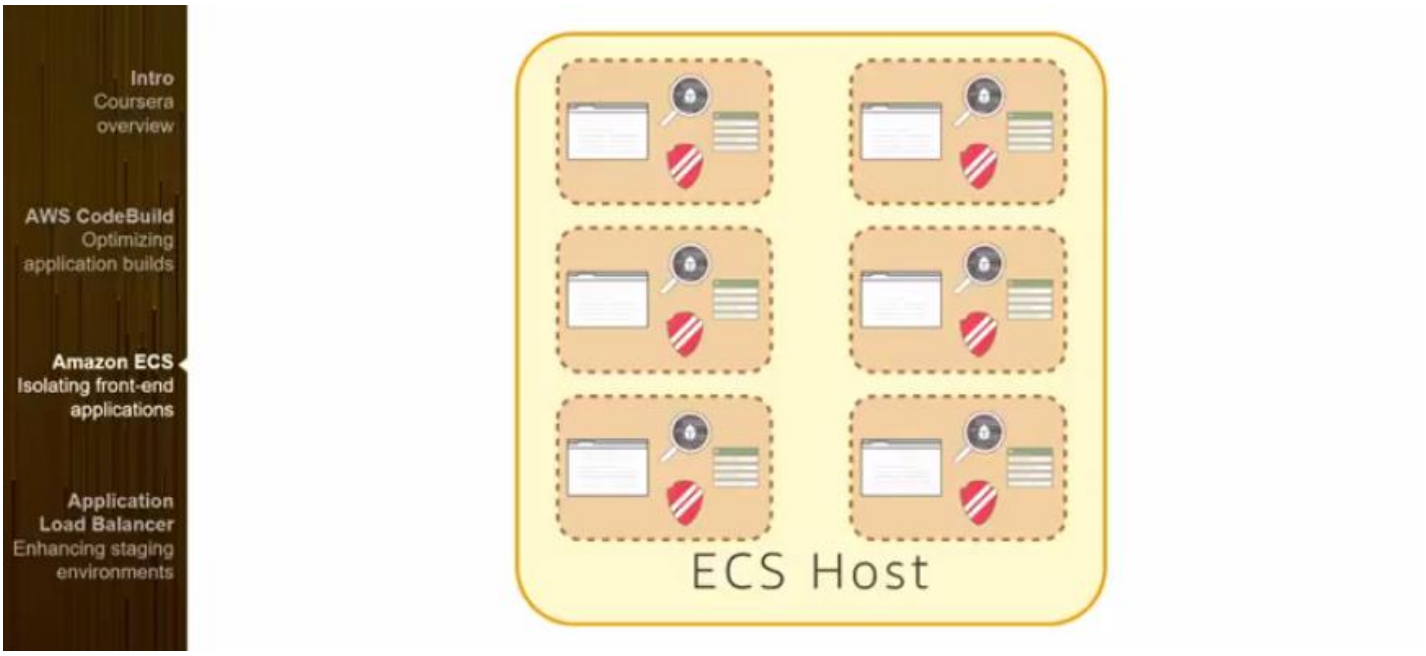
Amazon ECS
Isolating front-end applications

Application Load Balancer
Enhancing staging environments

Automate your deployments

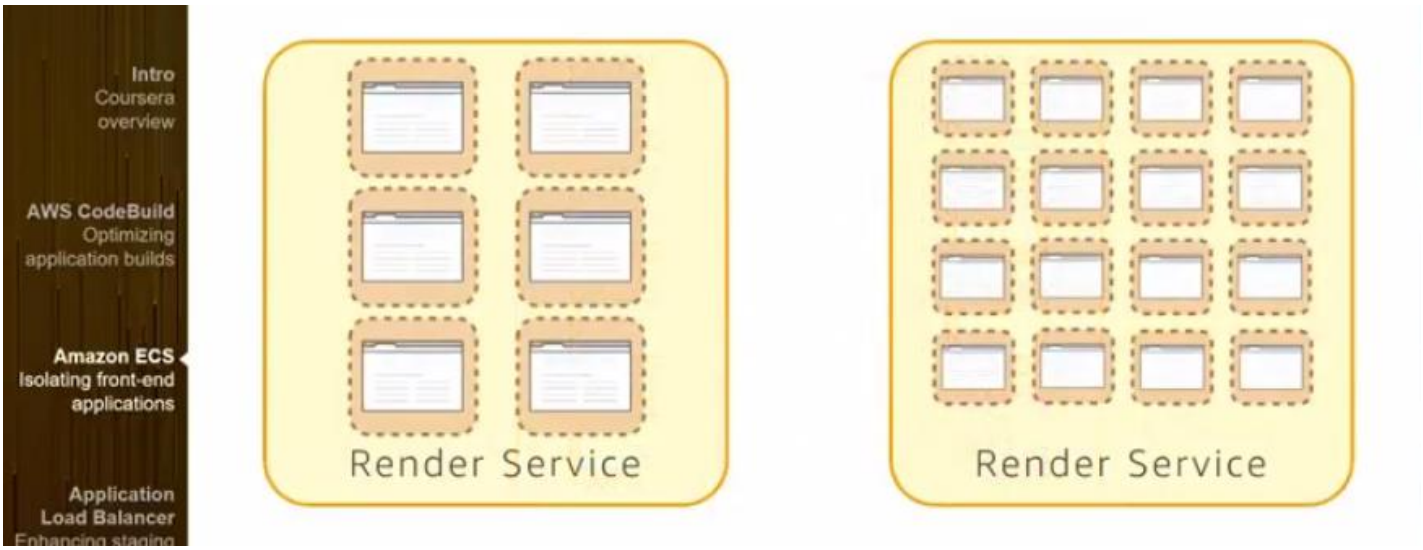
- Look into AWS CodeDeploy and AWS CloudFormation
- Use *services* for blue/green deployments

https://blog.codeship.com/easy-blue-green-
deployments-on-amazon-ec2-container-service/



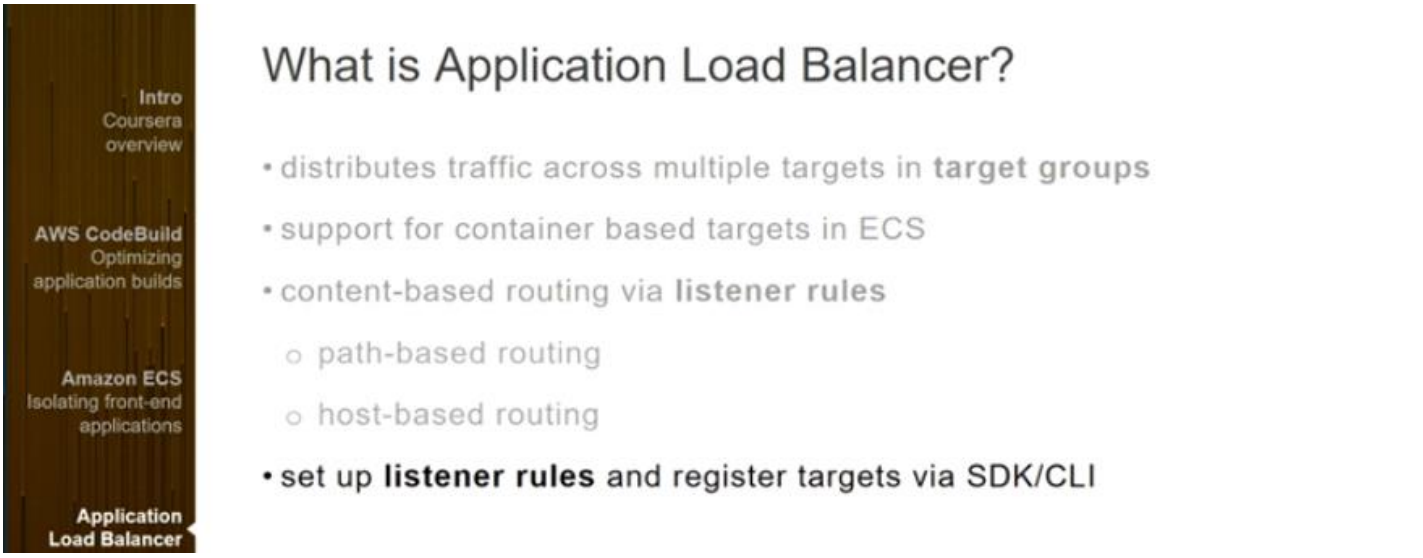
This is what we had earlier

Now we have all the utilities like SumoLogic in a separate container that logs get forwarded to from our actual apps

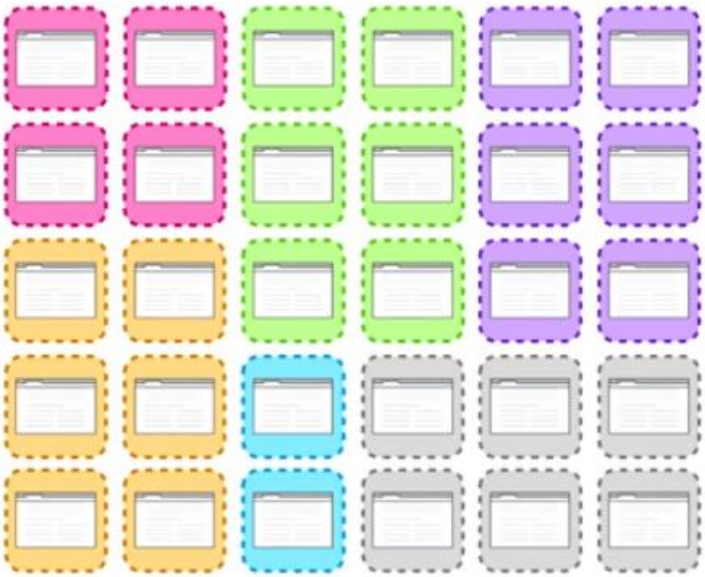


You might want to choose the correct EC2 instance types for your apps based on load tests against different instance types

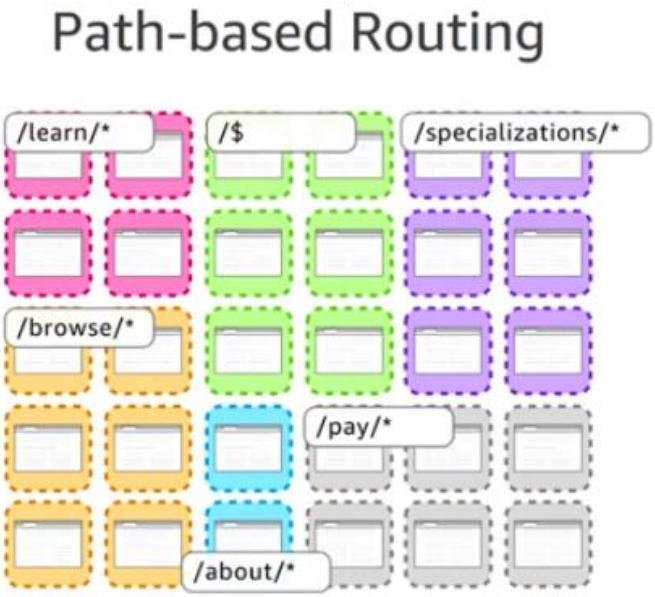
We need to route requests to different URLs running on coursera.org. this requires an ALB from AWS, it is part of the ELB ecosystem



We have to set up listener rules to help check and direct traffic



Since we have many different apps running in ECS, how do we route requests to correct app?



Path-based routing is too limited

- authentication can affect on routing
- experiments can affect routing

Host-based routing *can be* flexible

- An edge tier allows you to generate a Host header

Edge Tier

Given a request, figure out what application needs to serve the request.

Edge Tier

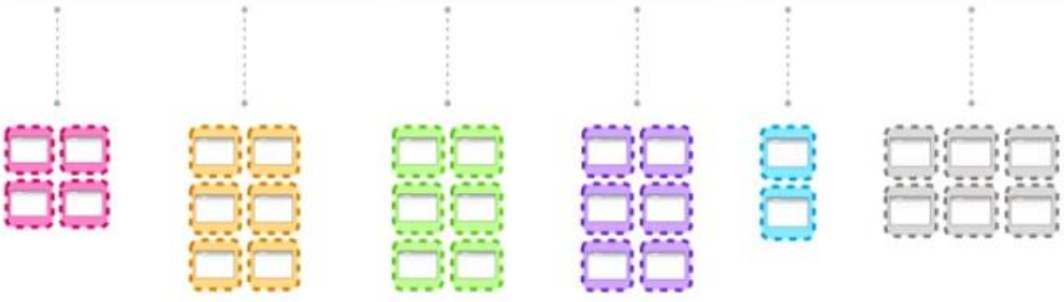
Takes advantage of Host Based Content Routing:
- Given a request, rewrite the Host Header "${appname}-${version}.coursera"
- Forward the request to ALB

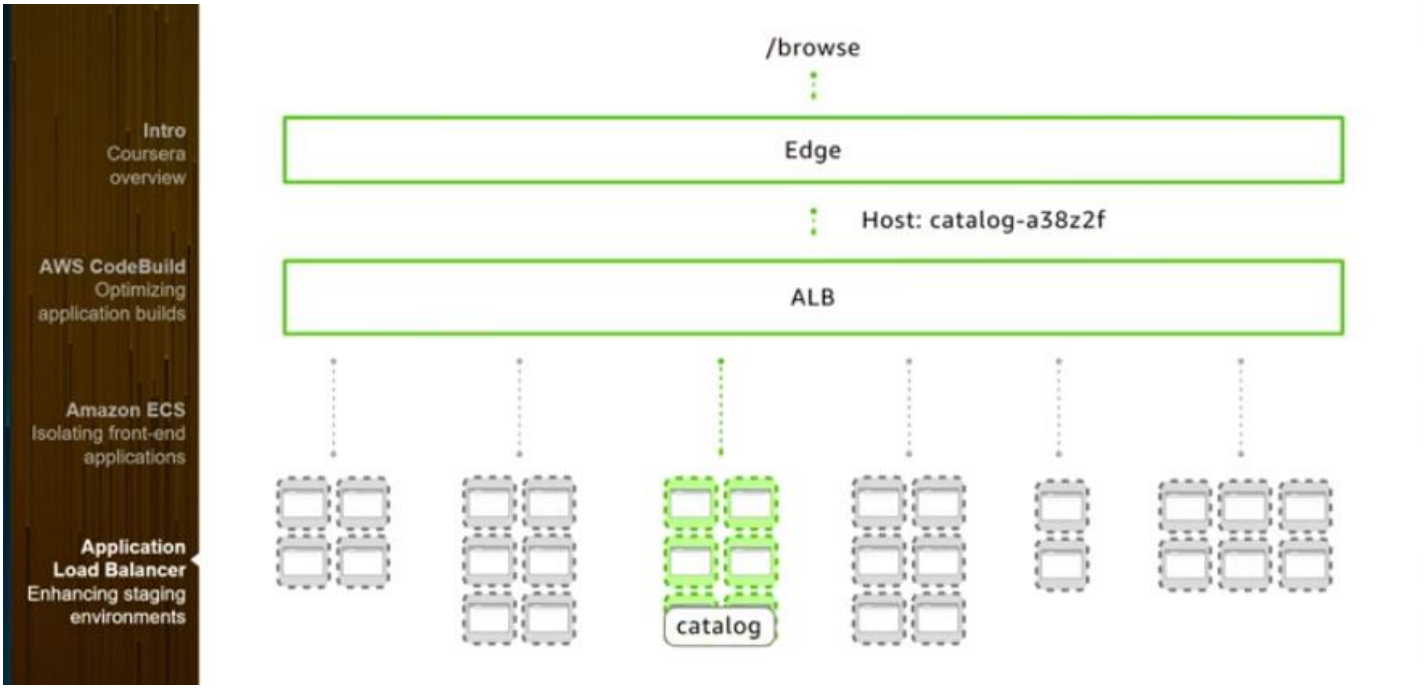
Edge

Application Load Balancer listener rules inspect the Host header to figure out to which target group it needs to forward the request

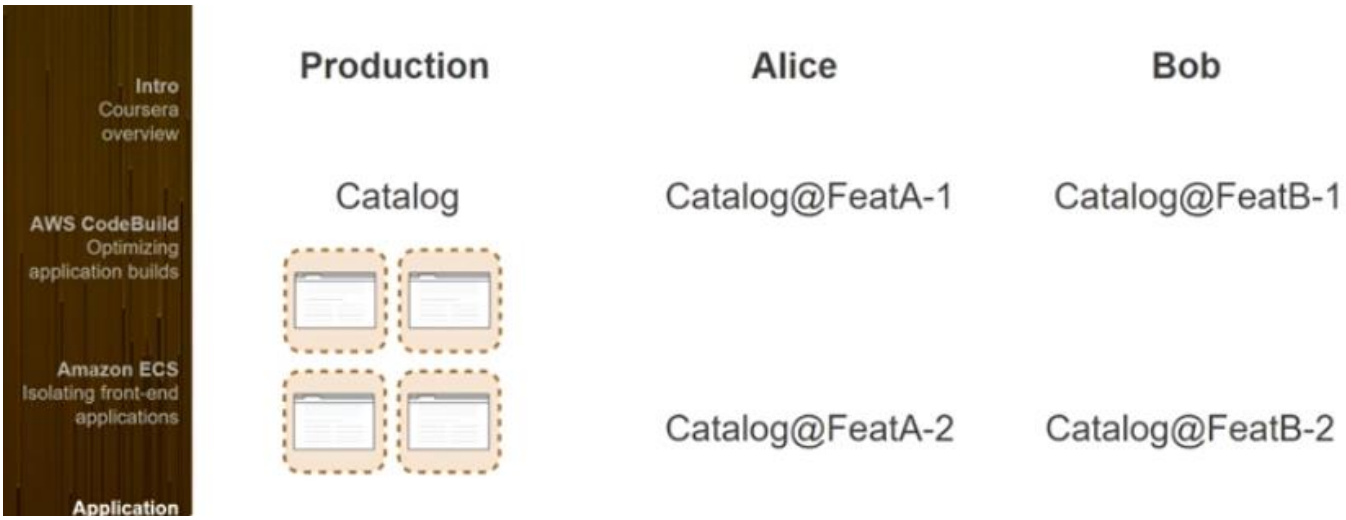
Edge

ALB

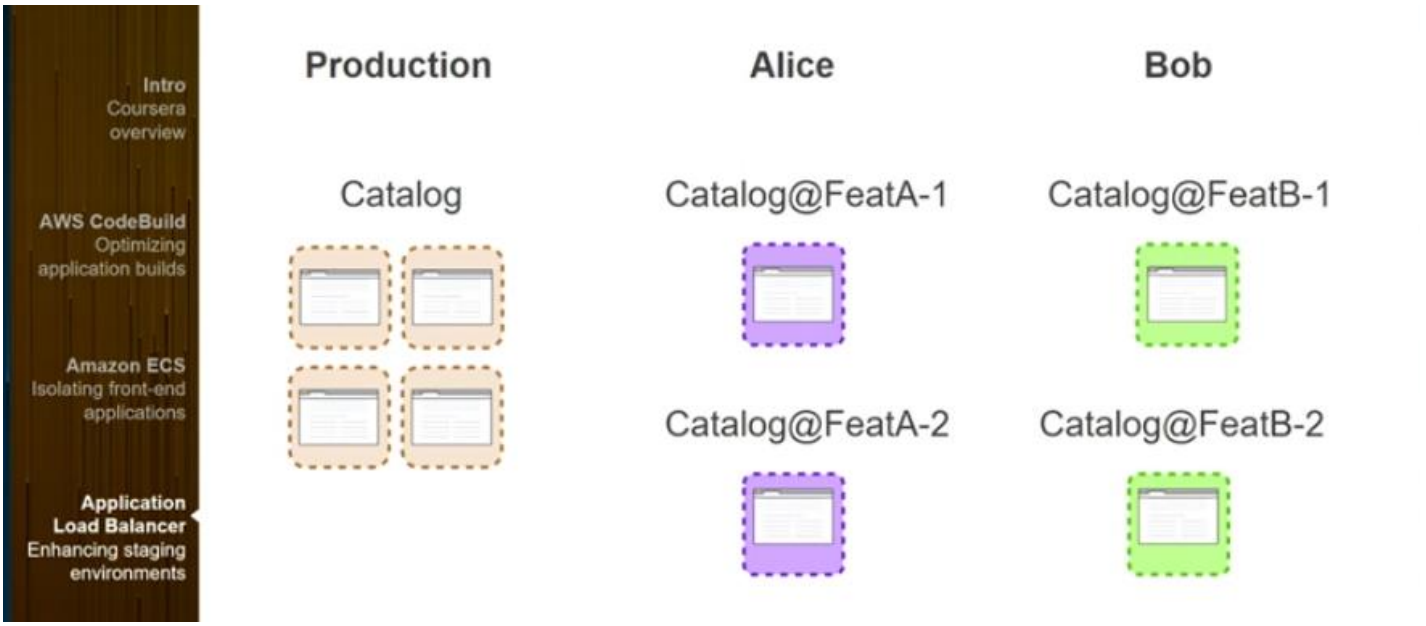




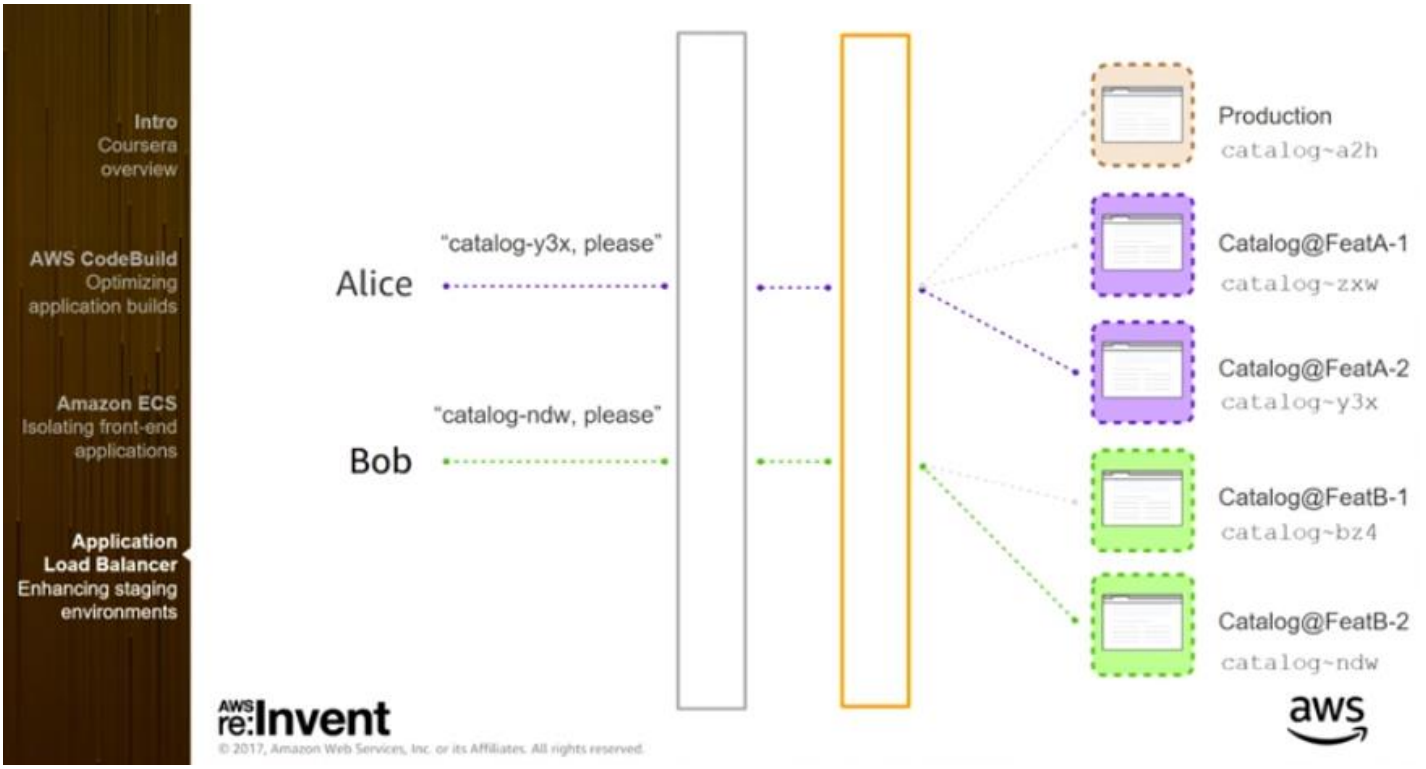
When a request comes in for the catalog service, it hits the Edge tire and that generates a host header based on the intended path. Then the edge tire passes the request along to the ALB, the ALB can then read the host header data and forward the request to the appropriate app. This is how we implement routing for our production services.



We can also preview changes too



If we change our build process to trigger on pull requests as well, we can then spin up containers running dev code also



We then have this type of scenario where we can have routing to the dev preview builds by setting up the correct host headers for the edge and the ALB in creating this temporary staging environments

Intro
Coursera overview

AWS CodeBuild
Optimizing application builds

Amazon ECS
Isolating front-end applications

Application Load Balancer Limitations

- Many hard limits exist, check documentation
  - Example: Max number of listener rules: 100
- Doesn't support traffic weighting
  - We implemented this in our edge tier

DevOps Lessons from Coursera:
Site Performance, Reliability, and Developer Productivity

Recap

Use AWS CodeBuild to scale your build environment on-demand

Pay only for what you use,
and take advantage of elasticity

Amazon ECS keeps your applications independent without the overhead of Amazon EC2

Reduce costs, and scale faster by containerizing your applications

ALBs give you the flexibility to route traffic in creative ways

Target specific hosts or containers with dynamic rules for on-demand environments


AWS re:Invent
Thanks for coming!
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws