

AWS Intensive

WELCOME!



John Kidd



A glowing lightbulb with a smiling filament. The filament is shaped like a wide, happy smile, and the bulb is illuminated with a warm, golden-yellow light. The background is dark, making the lightbulb stand out.

Container management



Managing docker containers

- Obviously containerization has become a major force in modern programming (if you're not using it I suggest that you look into it!)
- Containerization is basically a way to use Docker Containers to wrap code.
- Docker containers ARE a form of Infrastructure-as-code- that infrastructure coming in the form of **docker files** which we demonstrated in lesson one.



Container management

- So another big advantage of keeping your infrastructure-as-code is that you can have your developers create local environments based on infrastructure documents in the cloud.
- This can save you a TON of issues around re-creating the production environment for development and testing locally.
- If you've ever used the "Golden Image" to provision servers before you're familiar with the difficulties of maintaining the STATE of an environment for testing.



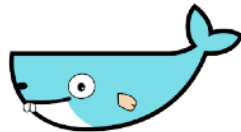
Container management (cont'd)

- So like any other codebase we would like to keep some sort of **version control** for our infrastructure code. This gives us a lot of the advantages of version control in our code base including things like keeping old versions of your infrastructure code to roll back to in case of an error.
- So ideally in this situation we would have some sort of repo similar to a github repository setup where we could store IMAGES from our DOCKERFILES.



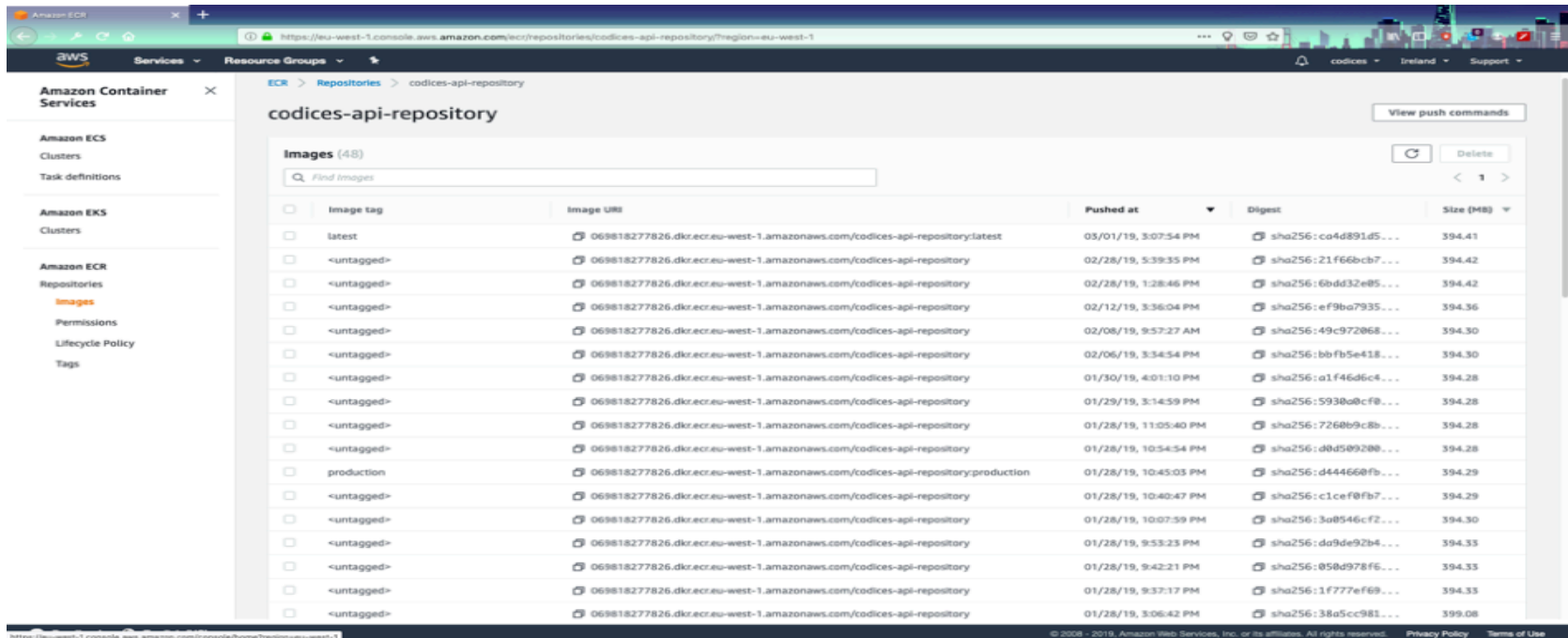
Container Management (continued)

- AWS provides a repository method for DOCKER IMAGES that your developers could use to develop from. You can push these images directly to production from the repository master branch and guarantee that the image in the repo (read- the environment) will match what's in production.
- The advantage here is also that you can pass an image down to your devs, they do work on the codebase, wrap up the code into an image, and pass it back and we can deploy that image directly.



Elastic Container Registry (ECR)

- So the **elastic container registry (ecr)** in AWS allows customers a **private registry** that will allow devs to pull, push, and deploy images while maintaining previous versions.



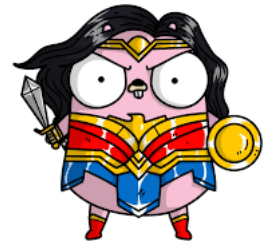
The screenshot displays the Amazon ECR console interface. The left sidebar shows the navigation menu with options like Amazon Container Services, Amazon ECS, Amazon EKS, and Amazon ECR. The main content area is titled 'codices-api-repository' and shows a list of 48 images. The table includes columns for Image tag, Image URI, Pushed at, Digest, and Size (MB).

Image tag	Image URI	Pushed at	Digest	Size (MB)
latest	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository:latest	03/01/19, 3:07:54 PM	sha256:ca4d891d5...	394.41
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	02/28/19, 5:39:35 PM	sha256:21f66bcb7...	394.42
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	02/28/19, 1:28:46 PM	sha256:6bdd32e85...	394.42
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	02/12/19, 3:36:04 PM	sha256:ef9ba7935...	394.36
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	02/08/19, 9:57:27 AM	sha256:49c972068...	394.30
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	02/06/19, 3:34:54 PM	sha256:bbfb5e418...	394.30
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/30/19, 4:01:10 PM	sha256:a1f46d6c4...	394.28
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/29/19, 3:14:59 PM	sha256:5938a8cf8...	394.28
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 11:05:40 PM	sha256:7268b9c8b...	394.28
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 10:54:54 PM	sha256:d8d509200...	394.28
production	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository:production	01/28/19, 10:45:03 PM	sha256:d444660fb...	394.29
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 10:40:47 PM	sha256:c1cef0fb7...	394.29
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 10:07:59 PM	sha256:3a0546cf2...	394.30
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 9:53:23 PM	sha256:da9de92b4...	394.33
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 9:42:21 PM	sha256:050d978f6...	394.33
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 9:37:17 PM	sha256:1f777ef69...	394.33
<untagged>	069818277826.dkr.ecr.eu-west-1.amazonaws.com/codices-api-repository	01/28/19, 3:06:42 PM	sha256:38a5cc981...	399.08



ECR Workflow

- So the Elastic Container Registry workflow is as follows:
 - DevOps pushes an initial base image to the repository
 - Developers pull the base image and do some work on the code.
 - Developers build (and locally run) the image that was pulled from the devops repo. Once they have proven that it works locally and it is ready to go...
 - Developers PUSH the updated image BACK to the repository with a tag that states the version.
 - Dev/OPS deploys the image to the **EC2 cluster**



Clusters in ECS

- So the next logical question is: where do we deploy the images to? WELL- AWS ECS comes with **CLUSTERS** that run the containers that you send.
- These **clusters** are ec2 instances that run your containers. The ec2 instances are **completely** run by AWS- you don't touch them or have anything at all to do with them. Your only concern is your containers.
- To set up this system you run your **container** as a **task** on a **service**



Clusters in ECS (continued)

- So this **service** is the logical hierarchy of **how** we want to run our containers. The **service** decides things like:
 - How many containers we want running at a base level
 - How many containers we want to expand to if containers start getting over N percent of CPU
 - What base image we want to use (basically which tag we want to use)
 - Any environment variables we want to add or overwrite in the container.



ECS Cluster management

- The biggest advantage of **service logic** is it's ability to *expand* and *contract* based on the load.
- For example: let's say you have some code that does transformation on a data stream. The incoming streaming data is, let's say, betting odds (that change constantly).
- As you can imagine the flow of this data would be extremely uneven....on slow times (with no matches) there would be very few odds changes while *during* matches it goes crazy.



ECS service management

- So your incoming demand for your service might look something like this- pretty uneven until a match...



ECS Service management

- So the **service** in the ECS cluster can have logic that says:
 - “When you get to 70% CPU utilization- go ahead and spin up another container to take some of that load. Follow this logic through between 2 and 50 containers. When you see the container demand going down...kill them off”.
- This (obviously) gives us a tremendous amount of elasticity in how we manage incoming demand on our AWS resources. This saves on COST and still leaves us **very little** to actually manage.



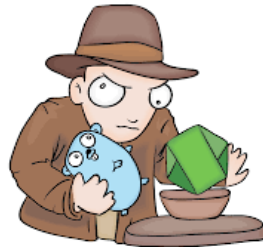
Create a cluster

- We're going to create a cluster together to show everyone how easy this is to do. For this exercise we're going to walk through deploying a basic API back end using another github category that I have created.
- Use this github link and the node file in module_05
<https://github.com/jwkidd3/deploy-to-fargate>



Deploying to Fargate

- So for this project we're going to use a very basic node.js express application that does nothing but give quotes from THE ROOM. The node application is written in node 11.
- cd into the module_05/fargateapp directory to see it and if you want to run it just type **node index.js**.
- Now let's deploy it to fargate! (We're going to do this without terraform for now just as an example).





Elastic Load Balancer



Elastic Load Balancer

- SO- we have our Tommy Wiseau quote app set up locally and deployed to fargate BUT...we still need a way to access it from the internet. Let's talk LOAD BALANCERS.
- So LOAD BALANCERS are basically just what the name implies: they manage incoming requests and send data to a **service**.
- We can make ECS into a **service** and direct traffic towards them fairly simply!



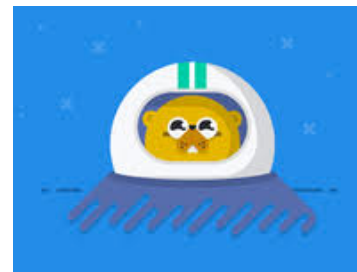
Elastic Load Balancer

- There are two types of elastic load balancers:
 - **Application Load Balancers:** Usually used for http/https communications **application load balancers** are intended, usually, for external communications with the internet. If you have a **service** (like our Tommy Wiseau quoting service) then you would use an ALB to manage the incoming http(s) requests and direct them to the appropriate container service.
 - **Network Load Balancers:** these work mostly in the TCP/TLS realm and deal with internal items to your application- not often from HTTP(s) – though the CAN be.



Load Balancers

- Load balancers come with some phenomenal advantages- like **health checks**- basically pings to your **service** that check to see if it is available. This means that you can default the ELB to send **404** in the case of a failed health check to your service.
- Load balancers aim for **target groups** which are really just synonyms for **services**.
- **ELBs** can be set up to accept incoming connections from domains you have purchased (though AWS **route 53** for example).



ECS set up as a target group

- ELBs can also **route** traffic through a really easy-to-use interface that allows you to send traffic to various places once it hits the ELB.
- You can define these routes and rules and therefore be set up to accept routes on sub-domains like /api/v1 for example.
- Check out the rules section for more on this.



Confused????

- GOOD! Ask Questions!!

