Initial environment setup

Overview

Welcome to catsndogs.lol, the fifth most highly rated cat and dog meme sharing website in Australia and New Zealand. Our mission is to serve a wide range quality of cat and dog memes to our customers. Memes come and go quickly, and we are starting to see larger and larger surges in customer demand.

catsndogs.lol uses Docker containers to host our application. Until today we’ve run everything on a spare laptop, but now we’re moving to the Amazon Elastic Container Service (ECS). Our DevOps Shepherd wants to take advantage of the latest and greatest features of the ECS platform. We also have several new initiatives that the developers and data science teams are keen to release.

As the new DevOps team, you will create the ECS environment, deploy the cats and dogs applications, cope with our hoped-for scaling issues, and enable the other teams to release new features to make our customers happier than ever.

**Welcome aboard!**

This document is a set of detailed instructions in case you need step-by-step help. The high level instructions which offer more freedom and a goal-oriented approach are at:

http://docs.catsndogs.lol

Prerequisites

This workshop requires:

* A laptop with Wi-Fi running Microsoft Windows, Mac OS X, or Linux.
* The AWSCLI installed.
* An Internet browser such as Chrome, Firefox, Safari, or Edge.
* An AWS account. You will create AWS resources including IAM roles during the workshop.
* An EC2 key pair created in the AWS region you are working in.

Initial Setup

1. Download the workshop materials zip from <http://docs.catsndogs.lol/materials.zip>  This contains the CloudFormation templates and other materials you will need during the workshop.
2. If you do not already have an EC2 keypair created, sign-in to the AWS EC2 console at <https://console.aws.amazon.com/ec2/>
   1. Click **Key Pairs** and then click **Create Key Pair.**
   2. Give the key pair a name and click **Create.** The console will generate a new key pair and download the private key.Keep this somewhere safe.
3. Deploy the initial CloudFormation template. This creates IAM roles, an S3 bucket, and other resources that you will use in later labs. The template is called **Lab0-baseline-setup.yml**.

In **Stack name**, enter **catsndogssetup.** Later labs will reference this stack by name, so if you choose a different stack name you will need to change the **LabSetupStackName** parameter in later labs.

1. Be sure to tick the **I acknowledge that AWS CloudFormation might create IAM resources with custom names** check box.

Task 1

Cost management and EC2 scaling

Overview

The catsndogs.lol environment has been running on a spare laptop, but today you will move everything to a new AWS ECS cluster.

Because the company is cost-conscious, the majority of our capacity will use EC2 Spot fleet instances. Because elasticity is also important, you will set up Auto Scaling for the Spot fleet to scale up and down as demand increases and decreases.

For long-term stability of core capacity, you will also add a small group of on-demand EC2 instances to the cluster.

At the end of this lab you will have an ECS cluster composed of Spot fleet instances with Auto Scaling enabled, an on-demand instance from an Auto Scaling group.

* 1. Create a new ECS cluster using Spot fleet
     1. Sign-in to the AWS management console and open the Amazon ECS console at [https://console.aws.amazon.com/ecs/](https://console.aws.amazon.com/s3/).
     2. In the AWS Console, ensure you have the correct region selected. The instructor will tell you which region to use.
     3. In the ECS console click **Clusters**, then click **Create Cluster.**
     4. In Cluster name, type **catsndogsECScluster** as the cluster name. This name is used in later labs. If you name the cluster something else you will have to remember this when running later commands.
     5. In **Provisioning Model** select **Spot.**
     6. Leave **Spot Instance allocation strategy** as **Diversified**.
     7. In **EC2 instance types** add several different instance types and sizes. We recommend you pick smaller instances sizes, such as:
* m4.large
* c4.large
* r4.large
* i3.large

You can also pick older generation families such as m3.large.

* + 1. In **Maximum big price (per instance/hour)** you can click the **Spot prices** link to view the current spot prices for the instance types and sizes you have selected. More information on how EC2 Spot instance pricing works is available on the Amazon EC2 Spot Instances Pricing page: <https://aws.amazon.com/ec2/spot/pricing/>
    2. Enter a maximum bid price. For the purposes of the workshop, $0.25 should offer an excellent chance of your Spot bid being fulfilled. It does not matter if your spot bid is not fulfilled. In a later step you will add an on-demand instance to the cluster.
    3. In **Number of instances** enter **3.**
    4. In **Key pair** select an existing EC2 Key pair for which you have the private key.
    5. In **VPC** select **ECSVPC**.
    6. In **Subnets** select all subnets containing the word **Private**.
    7. In Security group, select the Security Group containing the term **InstanceSecurityGroup.**
    8. In **Container Instance IAM role** select the IAM role containing the term **catsndogssetup-EC2Role.**
    9. In **IAM role for a Spot Fleet request** select the role with a name containing **catsndogssetup-SpotFleetTaggingRole**.
    10. Click **Create.**
    11. You will see the cluster creation steps appear. The final step is the creation of a CloudFormation stack. Note the name of this stack.
    12. Open the AWS console in a new browser tab and under **Management Tools**, click **CloudFormation.**
    13. Select the checkbox for the CloudFormation stack, and click the **Template** tab.
    14. The ECSSpotFleet resource has a Property named **LaunchSpecifications**, which contains **UserData**. This is about half way down the template.

**Note:** This script creates a Spot instance termination notice watcher script on each EC2 instance. That watcher script runs on each instance every two minutes. It polls the EC2 instance metadata service for a Spot termination notice. If the instance is scheduled for termination (because you have been outbid) the script sends a command to the ECS service to put itself into a DRAINING state. This prevents new tasks being scheduled on the instance, and if capacity is available in the cluster, ECS will start replacement tasks on other instances within the cluster.

More information about this script can be found on the AWS Compute blog: <https://aws.amazon.com/blogs/compute/powering-your-amazon-ecs-cluster-with-amazon-ec2-spot-instances/>

More information about the ECS DRAINING state can be found in the ECS documentation: <http://docs.aws.amazon.com/AmazonECS/latest/developerguide/container-instance-draining.html>

* 1. Set up Auto Scaling for the Spot fleet

In this task we will set up Auto Scaling for the Spot fleet, to provide cost-effective elasticity for the ECS Container Instances. Auto Scaling will use the ECS cluster MemoryReservation CloudWatch metric to scale the number of EC2 instances in the Spot fleet.

1. In the AWS Console **Management Tools** section click **CloudWatch.**
2. Click **Alarms**, then click **Create Alarm** to create an alarm for scaling out.
3. Click **ClusterName** under ECS Metrics.
4. Select the **MemoryReservation** metric for the cluster you created earlier, then click **Next**. It might take a minute or two for this new metric to appear in the CloudWatch console. If the metric is not yet listed, refresh the page and try again.
5. Give the alarm a name, for example **ScaleUpSpotFleet.**
6. Fill in the following under **Whenever: MemoryReservation:**

* Is: **>= 20**
* For: **2** consecutive period(s)

1. For the **Period** select **1 minute.**
2. For the **statistic** select **Standard, Maximum.**
3. In Actions, delete the pre-created Notification action.
4. Click **Create Alarm**.
5. Click **Create Alarm** to create the alarm for scaling in.
6. Click **ClusterName** under ECS Metrics.
7. Select the **MemoryReservation** metric for the cluster you created earlier, then click **Next**.
8. Give the alarm a name, for example **ScaleDownSpotFleet.**
9. Fill in the following under **Whenever: MemoryReservation:**

* Is: **<= 20**
* For: **2** consecutive period(s)

1. For the **Period** select **1 minute.**
2. For the **statistic** select **Standard, Maximum.**
3. In Actions, delete the pre-created Notification action.
4. Click **Create Alarm**.
5. Return to the AWS console home. In **Compute**, click **EC2.**
6. Click **Spot Requests.**
7. Select the checkbox by the Spot request.
8. Click the Auto Scaling tab in the lower pane, then click **Configure.**
9. In **Scale capacity between**, set **3** and **10** instances.
10. Under **Scaling policies**, click the **Scale Spot Fleet using step or simple scaling policies** option
11. In Scaling policies first update the ScaleUp policy:

* In **Policy Trigger** select the **ScaleUpSpotFleet** alarm you created earlier.
* Click **Define steps.**
* Click **Add step.**
* In **Modify Capacity**:
  + Add 2 instances when 20 <= MemoryReservation <= 50
  + Add 3 instances when 50 <= MemoryReservation <= infinity

1. Then update the ScaleDown policy:

* In **Policy Trigger** select the **ScaleDownSpotFleet** alarm you created earlier.
* Click **Define steps.**
* Click **Add step.**
* In **Modify Capacity**:
  + Remove 1 instances when 20 >= MemoryReservation > 10
  + Remove 2 instances when 10 >= MemoryReservation > -infinity

1. Click **Save**

More details on Auto Scaling for Spot fleet is available in the Spot Instances documentation: <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet-automatic-scaling.html>

* 1. Add an On-Demand Auto Scaling group to the cluster

In this task you will create an Auto Scaling group composed of two EC2 instances. If the Spot price goes above the maximum bid price, some or all of the Spot instances could be terminated. By using on-demand instances as well as Spot instances, you ensure the cluster will have capacity even if the Spot instances are terminated.

**Note:** For ECS clusters that will operate for a year or more, EC2 Reserved Instances provide both a capacity reservation and lower price per hour. We will not use Reserved Instances in this workshop but you should consider them for long-lived clusters.

If you have used Auto Scaling groups with ECS before, you can launch a CloudFormation stack that creates the resources below automatically. The CloudFormation template is called **Lab1-add-ondemand-asg-to-cluster.yml**. If you have not used Auto Scaling groups with ECS, you can follow the steps below to learn how to do this.

1. In the AWS Console **Compute** section click **EC2**, then click **Instances.**
2. Right click on an instance and click **Launch more like this.**
3. At the top of the console click **Choose Instance Type**
4. Select the m4.large instance type.
5. Click **Configure Instance**. If you receive a pop-up dialog, select “Yes, I want to continue with this instance type (m4.large)” and click **Next.**
6. Beside **Number of instances** click **Launch into Auto Scaling Group.**
7. In the pop-up dialogclick **Create Launch Configuration.** This launches the Auto Scaling Launch Configuration wizard and preserves the AMI and instance type and size.
8. In the **Configure Details** step enter **On-Demand-ECS** as the name.
9. In IAM role select the IAM role containing the term **EC2InstanceProfile.**
10. Expand Advanced Details. Copy the following text and paste it into the User data dialog box. This controls which ECS cluster the instance will join:

#!/bin/bash

echo ECS\_CLUSTER=catsndogsECScluster >> /etc/ecs/ecs.config

1. Click **Next: Add storage**.
2. Click **Next: Configure Security Group.**
   * 1. Click **Select an existing security group** and choose the Security Group containing the term **InstanceSecurityGroup.**
3. Click **Review** then click **Create launch configuration.**
4. In the pop-up dialog, select **Choose an existing key pair**, then select an EC2 key pair that you have the private key for. Click the checkbox and then click **Create launch configuration**.
5. The completes the Launch Configuration wizard and starts the Auto Scaling Group wizard. In Group name, enter **ECS-On-Demand-Group**.
6. In **Network**, select the **ECSVPC**.
7. In **Subnet** select all subnets containing the word **Private**. Click **Next: Configure scaling policies.**
8. Click **Review.**
9. Click **Create Auto Scaling group**, then click **Close.**
10. Return to the AWS Console and in the **Compute** section, click **EC2 Container Service**.
11. Click the ECS cluster **catsndogsECScluster**.
12. Click the **ECS Instances** tab and wait until the On-Demand instance appears in the list. You can continue the next task once the instance appears. If the instance does not appear within a few minutes, check the configuration of the Launch Configuration, specifically the **User data** script and the **VPC and subnet** selections.

You should now have an ECS cluster composed of three instances from the Spot fleet request, and one instance from the on-demand Auto Scaling group.

Lab 2

ECS Service deployment and task Auto Scaling

Overview

Now you have an ECS cluster running, you need to deploy the catsndogs.lol tasks and services. You also need to test the deployment works, and run a load test against the system to ensure it scales as expected.

You will deploy an ECS service for the homepage, and separate ECS services for cats and dogs. Having separate ECS services allows catsndogs.lol to scale the cats and dogs services separately based on demand.

You will set up Task Auto Scaling with proportional scaling actions. Multiple scaling actions allows ECS to respond by rapidly adding more tasks if the system comes under heavy load quickly.

Once the services and Auto Scaling are set up, you will launch a load generator that targets the cats and dogs pages. This will cause the services to scale up, which will also cause the Spot Fleet instances to scale up.

* 1. Create ECS Tasks and ECS Services for the homepage, cats, and dogs

This step will use CloudFormation to create the cats, dogs, and simplehomepage tasks and services within ECS, and associate the services with an Elastic Load Balancing Application Load Balancer. It will also create CloudWatch alarms for the cats and dogs services, which you will use to set up Task Auto Scaling.

1. In the AWS Console, ensure you have the correct region selected. The instructor will tell you which region to use.
2. In the **Management Tools** section click **CloudFormation.**
3. Click **Create Stack.**
4. Select **Upload a template to Amazon S3,** then click **Choose File** and choose the file named **Lab2-create-ecs-tasks-and-services.yml**
5. In Stack name, enter **catsndogsECStasksandservices**
6. Leave the ECSCluster and LabSetupStackName parameters at their default, unless you changed the name of the CloudFormation stack from the Lab setup, or named the ECS cluster something other than catsndogsECScluster.
7. Click **Next**, then click **Next** again, then click **Create.**
8. Wait until the stack status is **CREATE\_COMPLETE.**
9. Verify the catsndogs application works by loading the website:
   1. In the AWS Console, under **Compute** click **EC2.**
   2. Click Load Balancers.
   3. Copy the **DNS Name** of the load balancer with **catsn-catsn** in the name.
   4. Paste this into a new browser tab. You should see the catsndogs.lol homepage, and should be able to click the “I love cats” and “I love dogs” links to see pages served by the cats and dogs containers, respectively.
   5. Set up Task Auto Scaling for the cats and dogs services

In this task you will set up Task Auto Scaling for the cats and dogs services

1. In the **Compute** section click **EC2 Container Service.**
2. In the ECS console click **catsndogsECScluster** then the service with **Cats** in the name.
3. Click the **Update** button at the top right of the console.
4. On the **Configure Service** page click **Next Step.**
5. On the **Network configuration** page click **Next Step.**
6. On the Auto Scaling page select **Configure Service Auto Scaling to adjust your service’s desired count.**
7. Set **Minimum number of tasks** to 2.
8. Set **Desired number of tasks** to 2.
9. Set **Maximum number of tasks** to 100.
10. In **IAM role for Service Auto Scaling** select the role with **ECSAutoScaleRole** in the name.
11. Click **Add scaling policy** button.
12. In **Policy name** enter **CatsScaleUpPolicy**.
13. In **Execute policy when** select **Use an existing alarm** and choose the alarm with **CatsScaleUpAlarm** in the name**.**
14. In **Scaling action** clickthe **Add** button.
15. Enter: **Add 10 tasks** when **1000** <= RequestCount < **2000**
16. Enter: **Add 20 tasks** when **2000** <= RequestCount < **4000**
17. Click the **Add** button again.
18. Enter: **Add 25 tasks** when **4000** <= RequestCount < +infinity
19. Click **Save.**
20. Click **Add scaling policy** button.
21. In **Policy name** enter **CatsScaleDownPolicy**.
22. In **Execute policy when** select **Use an existing alarm** and choose the alarm with **CatsScaleDownAlarm** in the name**.**
23. In **Scaling action** clickthe **Add** button.
24. Enter: **Remove 10 tasks** when **1000** >= RequestCount > **100**
25. Enter: **Remove 5 tasks** when **100** >= RequestCount > -infinity
26. Click **Save.**
27. Click **Next step.**
28. Click **Update Service**.
29. Click **View Service**, then click the cluster name **catsndogsECScluster.**
30. Click the service with **Dogs** in the name.
31. Click the **Update** button at the top right of the console.
32. On the **Configure Service** page click **Next Step.**
33. On the **Network configuration** page click **Next Step.**
34. On the Auto Scaling page select **Configure Service Auto Scaling to adjust your service’s desired count.**
35. Set **Minimum number of tasks** to 2.
36. Set **Desired number of tasks** to 2.
37. Set **Maximum number of tasks** to 100.
38. In **IAM role for Service Auto Scaling** select the role with **ECSAutoScaleRole** in the name.
39. Click **Add scaling policy** button.
40. In **Policy name** enter **DogsScaleUpPolicy**.
41. In **Execute policy when** select **Use an existing alarm** and choose the **DogsScaleUpAlarm.**
42. In **Scaling action** click **Add** twice.
43. Enter: **Add 10 tasks** when **1000** <= RequestCount < **2000**
44. Enter: **Add 20 tasks** when **2000** <= RequestCount < **4000**
45. Enter: **Add 25 tasks** when **4000** <= RequestCount < +infinity
46. Click **Save.**
47. Click **Add scaling policy** button.
48. In **Policy name** enter **DogsScaleDownPolicy**.
49. In **Execute policy when** select **Use an existing alarm** and choose the alarm with **DogsScaleDownAlarm** in the name**.**
50. In **Scaling action** clickthe **Add** button.
51. Enter: **Remove 10 tasks** when **1000** >= RequestCount > **100**
52. Enter: **Remove 5 tasks** when **100** >= RequestCount > -infinity
53. Click **Save.**
54. Click **Next step.**
55. Click **Update Service**.
56. Click **View Service**, then click the cluster name **catsndogsECScluster.**
    1. Generate load and validate Task Auto Scaling works as expected

In this task, you will generate load to cause the cats and dogs services scale. As more cats and dogs tasks are added to the cluster, the MemoryReservation metric for the cluster will increase. Because the EC2 Spot fleet Auto Scaling is set up to scale based on MemoryReservation, this will cause the underlying EC2 Spot fleet to scale.

You will create a CloudFormation stack containing a load generator that sends load to the cats and dogs containers, and then verify the tasks scale as expected.

1. In the **Management Tools** section click **CloudFormation.**
2. Click **Create Stack.**
3. Select **Upload a template to Amazon S3,** then click **Choose File** and choose the file named **Lab2-loadgenerator.yml**
4. In Stack name, enter **catsndogslab2loadgenerator**
5. Leave the LabSetupStackName parameter at its default, unless you changed the name of the CloudFormation stack from the Lab setup.
6. Click **Next**, then click **Next** again, then click **Create.**
7. Wait until the stack status is **CREATE\_COMPLETE.**

Note: the LoadGenerator instance uses the Vegeta load generator. More information about this is available at: <https://github.com/tsenart/vegeta> . The CloudFormation template injects the URL of your load balancer so Vegeta sends requests to the correct endpoint

1. In the AWS Console, under **Management Tools** click **CloudWatch.**
2. Click **Metrics.**
3. On the **All metrics** tab, click **ApplicationELB**, then **Per AppELB, per AZ, per TG Metrics.**
4. Find the LoadBalancer where the name starts with **catsn-catsn** and select the **RequestCount** metrics.
5. On the **Graphed metrics** tab, change the **Statistic** to **Sum**, and the **Period** to **10 seconds**.
6. After a minute or two you should start to see an increase in request counts, to around 1500 each for the cats and dogs target groups. Note that the simpleHomepage target group is not accessed by the load generator.
7. Click **Alarms.**
8. After the load has been sustained for two minutes, the **Lab2-CatsScaleUpAlarm** and **Lab2-DogsScaleUpAlarm** should enter the ALARM state.
9. In the AWS Console, under **Compute** click **EC2 Container Service.**
10. In the ECS console click **Clusters**, then click the cluster **catsndogsECScluster**.
11. Click Services and click either the cats or dogs service.
12. Click the Events tab. You should see events as ECS adds more tasks to the Service.
    1. Validate the Spot fleet scales out

As more tasks are added to the cluster, the MemoryReservation metric will increase. Because the EC2 Spot fleet Auto Scaling is set up to scale based on MemoryReservation, this will cause the underlying EC2 Spot fleet to scale. In this task you will verify that Spot fleet Auto Scaling adds more EC2 instances to the cluster:

1. In the AWS Console, under **Management Tools** click **CloudWatch.**
2. Click **Alarms.**
3. Once sufficient copies of the cats and dogs tasks have started, the ScaleOut alarm you created in Lab 1 should change to ALARM state. Click this alarm and view the metric graph to see whether it has reached the alarm threshold.
4. Once it has reached the threshold and moved to ALARM state, move to the next step.
5. In the AWS Console, under **Compute** click **EC2.**
6. Click **Spot Requests** then select the Spot fleet request.
7. Click the **History** tab. You may see an **Event Type** of **autoScaling** with a **Status** of **pending**, otherwise you should see **Event Type** entries of **instanceChange** with a **Status** of **launched.**
8. In the AWS Console under the **Compute** section click **EC2 Container Service.**
9. In the ECS console click **catsndogsECScluster**
10. Click the **ECS Instances** tab.
11. Verify that the new instances are added to the cluster.
    1. Clean up

In this task, you will stop the load generator. As the load stops, the number of ECS tasks and number of instances in the Spot fleet will return to their default levels.

1. In the AWS Console, under **Compute** click **EC2.**
2. Click **Instances.**
3. Select the instance with **LoadGenerator** in the name.
4. Click **Actions** and select **Instance State**, then click **Stop.**

Lab 3

Deploying a new version of the cats service with secrets management

Overview

The development team at catsndogs.lol have been busy working on a new feature! The cats service will soon be able to serve up random unicorn pictures to lucky visitors. During the design process, it was decided that only the cats service should have access to the unicorns, and that the dogs service should not have access.

In order to accomplish this, the location of the unicorn images will be stored in an EC2 Systems Manager Parameter Store secure string. The new version of the cats task will run using an IAM role to enable access to the Parameter Store secure string. The dogs task will not use the IAM role, and so will not have access to the Parameter Store secure string.

In this lab, you will configure Parameter Store and deploy a new version of the cats task that can access the Parameter Store secure string.

* 1. Create secrets for the new version of the cats task

In this step, you will use EC2 Systems Manager Parameter Store to create a secure string for use with the new version of the cats container.

1. In the AWS Console, ensure you have the correct region selected. The instructor will tell you which region to use.
2. In the **Compute** section click **EC2**
3. At the bottom left of the page, click **Parameter Store.**
4. If you see the introductory page, choose **Get started**, otherwise click **Create Parameter.**
5. In **Name** enter **UnicornLocation**
6. In **Description** enter **Location of Unicorns for catsndogs ECS lab.**
7. In **Type** select **Secure String.**
8. In **KMS Key ID**, select **alias/keyForUnicorns (custom)**
9. In **Value**, enter **catsndogs-assets.s3.amazonaws.com**
10. Click **Create parameter**.
11. Click **Tags** tab and then click **Add Tags.**
12. For **Tag Key** enter **Classification.**
13. For **Tag Value** enter **Mythical.**

The tag information will be used to restrict access to the UnicornLocation parameter, more information can be found here:

<http://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-paramstore-access.html>

* 1. Deploy a new version of the cats task

In this step you will deploy the new version of the cats container, and pass in parameters so it can read the Parameter Store secure string you created.

1. In the AWS Console, ensure you have the correct region selected. The instructor will tell you which region to use.
2. In the **Compute** section click **ECS.**
3. Click **Task Definitions.**
4. Select the **cats** task and click **Create new revision.**
5. In **Task Role,** select the task role starting with **catsndogssetup-catsContainerTaskRole.**
6. Under **Container Definitions**, click the **cats** container name. This opens the container configuration window.
7. In **Image**, edit the container registry tag. Remove “:v1” and replace it with “:v2”.
8. Under **ENVIRONMENT** add two new **Env Variables.** The updated code in the new cats container will read these variables when starting.
   1. Key: **PARAMETER\_STORE\_NAME** Value: **UnicornLocation**
   2. Key: **REGION** Value: your region identifier, for example eu-west-1
   3. Key: **Tag** Value: **v2**
9. Click **Update.**
10. Click **Create.**
11. Note the revision number of the cats service that you just created. This will likely be cats:2 but may vary if you have done this lab before.
12. Click **Clusters** and then click **catsndogsECScluster.**
13. Select the **cats** service and click **Update.**
14. In **Task Definition** select the revision of the cats task that you noted in step 11.
15. In **Minimum healthy percent** enter **50.**
16. In **Maximum healthy percent** enter **100.**

Note: The update to the cats Service will replace the containers that make up the service. ECS offers you control over how the replacement process works. Because the cats containers are serving production traffic, you should not stop all the containers before starting new ones. By specifying a Minimum health percent of 50 and a Maximum healthy percent of 100, ECS will terminate up to 50 percent of the active cats containers, then start new containers. Once the new containers are healthy, ECS will terminate the remaining 50 percent of and replace those. This way, the cats service does not exceed its current footprint.

The default values, a Minimum healthy percent of 100 and Maximum healthy percent of 200, would briefly double the number of cats containers during deployment. That may be acceptable in many situations, however our deployment strategy is not to exceed the current container count.

1. Click **Next step** until you reach the end of the wizard, then click **Update service.**
2. Click **View service**. The Deployments tab should show the PRIMARY deployment as well as the ACTIVE deployment.
3. Click the **Events** tab. Depending on the number of cats tasks that were running at the time you updated the service, ECS will show events terminating groups of cats tasks, and starting groups of the new revision.
4. Click the **Tasks** tab. You should see tasks with the **Last status** of RUNNING and the **Task Definition** of the revision number you noted in step 11.
5. In the AWS Console, under **Compute** click **EC2.**
6. Click Load Balancers.
7. Copy the **DNS Name** of the load balancer with **catsndogssetup** in the name.
8. Paste this into a new browser tab. You should see the catsndogs.lol homepage
9. Click the “I love cats” link.
10. You should see the cats page change to the “new and improved v2 release” page with a blue background. There is a one in ten chance that the page will load a unicorn image. Shout out loud when you see one!

**Extension activity:** The new cats pages show the containerID at the bottom of the page. Examine the cats\_v2 source code and work out how this information is obtained, and how the v2 cats container obtains the location of the unicorns from Parameter Store.

Lab 4

Running ECS tasks based on time and events

Overview

catsndogs is growing and becoming more successful, but rapid growth brings its own problems. Someone (probably Buzzy) has uploaded several cat images that haven’t been through our rigorous assessment process.

In response, the development team have created a new automatic image assessment algorithm called ImageAssessor. The initial release selects several images at random, removes them, and then exits. A future release will select identify and remove only non-cat images. The priority now is to get the ImageAssessor container into production.

The cat-image-standards sub-committee has determined that running the ImageAssessor container every two minutes should ensure our quality bar remains high.

You will need to create a new ECS Task for the ImageAssesssor, and create a scheduled ECS task, which runs the container on a regular schedule.

Once the ImageAssessor has removed some images from the cats containers, you will run override the environment variables of the ImageAssessor container to reset the cats images.

* 1. Create a new ECS Task for the ImageAssessor container

In this task you will create a new Task definition that will run the image assessments.

1. In the AWS Console, in the **Compute** section click **ECS.**
2. Click **Task Definitions.**
3. Click **Create new Task Definition.**
4. In **Task Definition Name\*** enter **ImageAssessor**
5. Under **Container Defintions** click **Add Container**
6. In **Container Name** enter **ImageAssessmentContainer**
7. In **Image** enter **205094881157.dkr.ecr.us-west-2.amazonaws.com/image-assessor:latest**
8. **In Memory Limits (MiB)** enter **128.**
9. In **Env Variables** you need to enter the URL of the catsndogs load balancer. The ImageAssessor container uses this to send API commands to the cats containers:
   1. Key: ALB\_URL
   2. Value: <URL of the load balancer> for example: http://catsn-catsn-123455678-abcdefgh.us-west-2.elb.amazonaws.com
10. Click **Add.**
11. Click **Create.**
    1. Create a scheduled ECS task.

In this task you will create a scheduled ECS task which executes every five minutes:

1. In the AWS Console, in the **Compute** section click **ECS.**
2. Click the cluster **catsndogsECScluster.**
3. On the **Scheduled Tasks**, click **Create**.
4. In **Create scheduled task**:
   1. In **Scheduled rule name\***, type **ImageAssessor**.
   2. For **Scheduled rule type**, choose **Run at fixed interval**.
   3. For **Run at fixed interval\*,** enter **2**, and from the drop list, select **Minutes**.
5. In Scheduled target:
   1. In **Target id\***, enter **catsndogsreinvent2017**.
   2. For **Task Definition**, from the drop list, choose the **ImageAssessor:1** image.
   3. Set the **Number of tasks\*** to **1**.
   4. For **CloudWatch Events IAM role for this target**, choose the role with **catsndogssetup** in the name.
6. Click **Create**
   1. Verify the Image Assessor is working
7. Once the schedule ECS task is created, click the **ImageAssessor** task, list on the **Scheduled Tasks** tab.
8. Click **View CloudWatch metrics**.
9. Until the task has run at least once you may see the following text. Wait a minute and refresh the page.

Your search - ImageAssessor - did not match any metrics.

Tips:

Tags such as EC2 instance name tags are not supported in metric search.

Make sure that all words are spelled correctly.

Try different keywords.

Try fewer keywords.

1. Select the Invocations and TriggeredRules metrics when they become available. Ensure the Invocations count is 1.
2. In your web browser, open the load balancer URL and click on the “I love cats” link. You should see pages with cat pictures missing as the Image Assessor removes pictures.
3. You can verify which pictures remain by querying the cats API. Replace the URL in the example below with the URL of your load balancer:

<http://catsn-catsn-123455678-abcdefgh.us-west-2.elb.amazonaws.com/cats/api/list-pictures/>

You should see a JSON document listing the pictures that remain in the container, for example: {"2.jpg": "true", "10.jpg": "true", "7.jpg": "true}

If many cats containers are running, the ImageAssessor may not have removed images from all of them. Refresh your browser to connect to a different container and view the list of images in that container. You may want to allow the ImageAssessor to run two or three times to remove at least some images from every container before continuing.

1. In the AWS Console, in the **Compute** section click **ECS.**
2. Click the cluster **catsndogsECScluster**
3. **.** On the **Scheduled Tasks**, click the **ImageAssessor** task.
4. Click the **Edit** button in the top right of the screen.
5. Uncheck the **Schedule rule enable\*** check box, to disable the rule.
6. Click **Update**.
   1. Reset the cats images by overriding an environment variable in the ImageAssessor task

The ImageAssessor can also reset all of the cats image if the following environment variable is set for the task definition: RESETPICTURES: 1

1. In the ECS Console, click the **catsndogsECScluster.**
2. Click the **Tasks** tab and then click **Run new Task.**
3. In **Task Definition** select the most recent revision of the **ImageAssessor** task.
4. In **Cluster** select the **catsndogsECScluster**.
5. In **Number of tasks** enter **2**.
6. Leave **Task Group** blank.
7. Expand **Advanced Options**.
8. Under **Container Overrides** expand the **ImageAssessor** container.
9. In **Environment variable overrides** click the + to add a new environment variable.
10. In Key enter **RESETPICTURES** and in Value enter **1**
11. Click **Run** **Task.**
12. In the **Tasks** tab the **ImageAssessor** tasks should move appear with a **Last status** of PENDING. In a few seconds this will change from PENDING to RUNNING.
13. The tasks will run for 30 seconds and then exit.
14. Once they have exited, click **Desired task status: Stopped**
15. Find one of the ImageAssessor tasks in the list and click the **Task** identifier.
16. Under Containers, expand the image-assessor container. You should see the **Exit code0** indicating the container exited successfully.
17. Verify the cats pictures have been reset by querying the cats API. Replace the host in the example below with the URL of your load balancer:

<http://catsn-catsn-123455678-abcdefgh.us-west-2.elb.amazonaws.com/cats/api/list-pictures/>

Lab 5

Machine Learning containers and placement constraints

Overview

After the quite simplistic image filtering using the ImageAssessor container, the catsndogs.lol Data Scientists want to deploy a machine learning container. This should be much better at identifying cats (and dogs!) in the images.

However, they only want to run it on EC2 instances with a large number of CPUs so it doesn’t interfere with the website.

In this lab, you will create a new task and configure an ECS custom constraint that uses built-in attributes. You will then create a new service with a custom placement strategy for the tasks within the service. This ensures the tasks are scheduled on container instances that meet the data science team’s requirements.

After completing this lab, you will understand how to use ECS placement constraints to schedule tasks on specific container instance types, and attach custom attributes to container instances, then use those attributes to constrain the placement of tasks.

* 1. Create a new task definition for the MXNet container

In this step, you will create a new task definition for a deep learning container running the MXNet framework, with a placement constraint to ensure tasks run only on certain instance types.

1. Sign-in to the AWS management console and open the Amazon ECS console at [https://console.aws.amazon.com/ecs/](https://console.aws.amazon.com/s3/).
2. Select **Task Definitions** from the left-hand menu.
3. Click **Create new Task Definition**.
4. In **Task Definition Name**, enter **mxnet**.
5. Click **Add Container**.
6. In the **Add container** dialog, under Standard:
   1. In **Container name**, enter mxnet
   2. In **Image**, paste the repository URI and add the latest tag.

205094881157.dkr.ecr.us-west-2.amazonaws.com/mxnet:latest

* 1. In **Memory Limits (MiB)**, set **Hard Limit** to **2048**.
  2. Click **Add**.

1. In **Constraint**, click **Add constraint**.
2. Set the **Expression** to use an instance type that is currently running in the cluster. For example if one of the instance types was an c4.large, you would enter:

attribute:ecs.instance-type == c4.large

To check the instance types running, open the Clusters view in a new tab, click **catsndogsECScluster** and select the **ECS Instances tab.**

From the pop-up window, click the cog button, , and select **ecs.instance-type.**

Scroll along the list of instances to see the **ecs.instance-type** value**.**

1. Click **Create**.
   1. Create a new service for MXNet with a custom placement strategy

In this step, you will create a new ECS Service that will ensure two instances of the MXNet container run at all times.

1. In the navigation pane click **Clusters**.
2. Click the cluster **catsndogsECScluster**.
3. On the **Services** tab, click **Create**.
4. In Configure service:
   1. In **Task definition**, choose **mxnet:1**
   2. In **Cluster**, choose **catsndogsECScluster**.
   3. In **Service name**, enter **mxnetservice**
   4. In **Number of tasks**, enter **2.**
   5. Leave **Minimum healthy percent** and **Maximum percent** at their defaults.

You will now use a custom placement template to force the MXNet tasks to spread across Availability Zones, then across different instance types, and then BinPack based on Memory:

1. In **Placement Templates** select **Custom.**
2. In **Type** choose **Spread,** and in **Field** choose **attribute:ecs.availability-zone**
3. Click **Add Strategy.**
4. In **Type** choose **Spread,** and in **Field** choose **attribute:ecs.instance-type**
5. Click **Add Strategy.**
6. In **Type** choose **BinPack,** and in **Field** choose **Memory**
7. Click **Next step**.
8. In **Network configuration**, for **Load Balancing**, choose **ELB Type** of **None**.
9. Click **Next step**.
10. In **Auto Scaling (optional)**, for **Service Auto Scaling**, choose **Do not adjust the service’s desired count**.
11. Click **Next step**.
12. Review the settings, and click **Create Service**.
13. Click **View Service**.
    1. Ensure the placement constraints are being honored

In this step, you will ensure that the constraint you configured for the mxnet task is being honored by the ECS service scheduler.

1. In the navigation pane click **Clusters**.
2. Click the **catsndogsECScluster**.
3. Click the **Tasks** tab and locate one of the **mxnet** tasks from the list of running tasks.
4. Click on the **Task** ID.
5. In the **Details** view, locate and click on the **EC2 instance id**.
6. The EC2 console will open and display the container instance.
7. Check the **Instance Type**, it should be the type and size you selected in the earlier steps.
   1. Add a container instance custom attribute using the AWS CLI, to control task placement for the cats service

In addition to the built-in attributes of instance type, AMI, availability zone and operating system type, you can also constrain the placement of tasks using custom attributes. A custom attribute is metadata added to container instances. Each attribute has a name, and an optional string value.

Management have asked that we enforce strict segregation between the cats and the dogs to stop the fighting with each other. In this task, you will use the AWS Management console to add a custom attribute to a container instance. The custom attribute will then be used to constrain the cats containers to a specific container instance.

In this step, you will use the AWS CLI to add a custom attribute to a container instance. You will then update the cats task to add a constraint using the custom attribute you created.

1. Open to the AWS management console and open the Amazon ECS console at [https://console.aws.amazon.com/ecs/](https://console.aws.amazon.com/s3/).
2. In the navigation pane click **Clusters**.
3. Click the cluster **catsndogsECScluster**.
4. Click the **ECS Instances** tab and copy a **container instance ID** from the **Container Instance** column.
5. This step can be completed from your laptop. From the command prompt, run the following AWS CLI command. Replace <region> with your region, and <container\_instance\_id> with the instance ID you copied in the previous step:

aws ecs put-attributes --cluster catsndogsECScluster --attributes "name=catslike,value=catnip,targetType=container-instance,targetId=<container\_instance\_id>" --region <your-region-name>

1. You should see a response containing details of the attribute.
2. In the navigation pane choose **Task Definitions**.
3. Select **cats** from the list of task definitions.
4. Click **Create new revision**.
5. In **Constraint**, click **Add constraint.**
6. Set the **Expression** to:

attribute:catslike == catnip

1. Click **Create**.

Note: You can also try experimenting with some of the built-in attributes like **instance type, AMI, availability zone** and **operating system type**.

Verify that the custom attribute you created is visible in the console:

1. Open to the AWS management console and open the Amazon ECS console at [https://console.aws.amazon.com/ecs/](https://console.aws.amazon.com/s3/).
2. In the navigation pane click **Clusters**.
3. Click the cluster **catsndogsECScluster**.
4. Click the **ECS Instances** tab and select the check box for the container instance you added the custom attribute to.
5. Click **Actions** and **View/Edit Attributes**.
6. Verity the **Catslike** key exists and the value is **Catnip.**
7. Click **Close** to return to the ECS Instances tab.
   1. Update the cats service to use the custom attribute

In this step, you will update the cats service to use the new task definition you created in the previous step.

1. In the navigation pane choose **Clusters**.
2. Click the cluster **catsndogsECScluster**.
3. Click the **Services** tab then click the cats service.
4. Click **Update**.
5. In Configure service:
   1. In **Task definition**, choose the task definition you created in the earlier step.
   2. In **Cluster**, choose the cluster **catsndogsECScluster**
6. Click **Next step.**
7. In **Load Balancing**, choose **Next step**.
8. In **Service Auto Scaling (optional)**, click **Next step**.
9. Review the settings, and click **Update service**.
10. Click **View Service**.
    1. Ensure the placement constraints are being honored

In this step, you will use the AWS management console to ensure that the constraint you configured for the cats task, is being honored by the ECS service scheduler.

1. In the navigation pane choose **Clusters**.
2. Click the cluster **catsndogsECScluster**.
3. Click the **ECS Instances** tab and locate the instance with the **Container Instance** you added the custom attribute to in the earlier step.
4. You should see that all **cats** tasks are now running on the container instance with the **catslike** attribute.

Lab 6

Automated Deployments

Overview

The catsndogs.lol development team are planning to release updates to their applications more frequently. They want to build an automated deployment pipeline, that can be used to deploy updated versions of their applications with minimal manual intervention, to reduce the time it takes to get exciting new capabilities in the hands of their users.

In this lab, you will set up AWS CodePipeline to monitor an S3 bucket for changes to the source code. When new source code is uploaded to the bucket, CodePipeline will coordinate building and deploying the Docker based application.

You will create an AWS CodeBuild project build the Docker image and push it to a repository. The CodeBuild project will tag the newly built dogs containers with a version number.

You will also integrate AWS CodePipeline with AWS CloudFormation to update the existing ECS tasks and services. The pipeline will use the version number of the dogs containers as a parameter when updating the CloudFormation stack, so that the right version of the container is deployed.

* 1. Upload the source code to an S3 bucket

In this task, you will upload two artifacts to an S3 bucket. These artifacts are used as inputs to the automated deployment pipeline that you will create in later sections.

1. Sign-in to the AWS management console.
2. In the **Storage** section click **S3**.
3. Click on the S3 bucket with **CodeUploadS3Bucket** in the name.
4. Click **Upload**.
5. Click **Add files**.
6. From the Lab-6-Artifacts folder, upload **templates.zip** and **dogs.zip.**
7. On **Set permissions**, click **Next**.
8. On **Set properties**, click **Next**.
9. On **Review**, click **Upload**.

**Note:** Make a note of the full name of S3 bucket with **CodeUploadS3Bucket** in the name. You will need this in the following section.

The dogs.zip file contains all the required components to build the dogs container. The directory structure of the dogs.zip:

*(root directory name)*  
 *|-- buildspec.yml  
 `-- Dockerfile  
 `-- nginx.conf  
 `-- index.html*

The templates.zip file contains the CloudFormation template that was used to create the cats, dogs and simplehomepage tasks and services in Lab 2. The directory structure of the templates.zip:

*(root directory name)*  
 *|-- Lab2-create-ecs-tasks-and-services.yml*

* 1. Create an AWS CodePipeline pipeline

In this step, you will create a new AWS CodePipeline pipeline that you will use to orchestrate the deployment the new version of the Dogs application to your Amazon ECS cluster.

1. Sign-in to the AWS management console and open the AWS CodePipeline console at <https://console.aws.amazon.com/codepipeline/>.
2. If you see the introductory page, choose **Get started**, otherwise, choose **Create pipeline**.
3. In **Step 1: Name**, in **Pipeline name**, type **CatsnDogsPipeline**, and then click **Next step**.
4. In **Step 2: Source**, in **Service provider**, choose **Amazon S3**. In **Amazon S3 location**, type the name of the S3 bucket with **CodeUploadS3Bucket** in the name. This was created by the CloudFormation stack you deployed at the start of the workshop. Append /dogs.zip and click **Next step**.

Example: s3://CodeUploadS3Bucket-mitch/dogs.zip

1. In **Step 3: Build**, choose **AWS CodeBuild.**
2. Under **Configure your project** choose **Create a new build project**.
3. Name your project **CatsnDogsBuild.**
4. Under **Environment: How to build**:
   1. In **Environment image**, choose **Use an image managed by AWS CodeBuild**.
   2. In **Operating system**, choose **Ubuntu.**
   3. In **Runtime**, choose **Docker.**
   4. In **Version,** choose **aws/codebuild/docker:1.12.1.**
   5. In **Build specification,** choose **Use the buildspec.yml in the source code root directory.**

**Note:** The buildspec.yml is one of the files contained within the dogs.zip file that you uploaded to S3.

1. For **Service Role,** select **Choose an existing service role from your account,** use IAM the role with **CatsnDogsBuild** in the name.
2. **Expand Show advanced settings**:
   1. In **Environment variables** add the following:
      1. **AWS\_DEFAULT\_REGION**: <your AWS region>
      2. **AWS\_ACCOUNT\_ID**: <the account ID of your AWS account>
      3. **REPOSITORY\_URI**: <URI of your dogs ECR repository> for example: 12345567891011.dkr.ecr.ap-southeast-2.amazonaws.com/dogs

**Note:** The URI of your dogs repository is listed in the ECS Console. Click on Repositories, then on the dogs repository.

1. Click **Save build project**.
2. Click **Next step**.
3. In **Step 4: Deploy**, in **Deploy**:
   1. For **Deployment provider**, choose **No Deployment.**

**Note:** The deployment configuration requires additional settings beyond those available through the wizard. You will add a custom Deploy stage in the following sections.

1. In **Step 5: Service Role**, **in Role name\*** choose the IAM role with **CatsnDogsPipeline** in the name and click **Next step**.
2. Review the settings and click **Create pipeline**.

**Note:** In the following steps, you will make some changes to the pipeline you created.

Because you deployed ECS tasks and services to your cluster using CloudFormation, you will continue to use that as the deployment tool. CloudFormation will deploy updates to the running tasks and services by way of a stack update.

In order to use the CloudFormation template as part of the pipeline, the template needs to be defined within the pipeline. You will configure an additional action within the pipeline’s Source stage to download the template from the S3 bucket with CodeUploadS3Bucket in the name.

Parameter overrides will be used to update parameter called DogTag within the CloudFormation template with the new version of the Docker images created during the build process.

1. Click **Edit**.
2. Within the pipeline editor, click the edit icon  for the **Source** stage.
3. Add a new **Action:**
   1. For Action Category, choose **Source**.
   2. For **Action name**, enter **Template**
   3. For **Source provider**, choose **Amazon S3**.
   4. For Amazon S3 location, choose **Amazon S3**. In **Amazon S3 location**, type the name of the S3 bucket with **CodeUploadS3Bucket** in the name. This was created by the CloudFormation stack you deployed at the start of the workshop. Append /templates.zip, for example:

s3://CodeUploadS3Bucket-mitch/templates.zip

1. In **Output artifacts**, enter **template**
2. Click **Add action**.
3. Below the **Build** stage, click the add stage icon,, to add a new stage to the pipeline.
4. Name the stage **Deploy**.
5. Add a new **Action**:
   1. For **Action Category**, choose **Deploy**.
   2. For **Action name**, enter **deploy-dogs.**
   3. For **Deployment provider**, choose **CloudFormation**.
6. In **AWS CloudFormation:**
   1. For **Action mode**, choose **Create or update a stack**
   2. For **Stack name**, choose the stack you created in Lab 2 to deploy the ECS tasks and services. If you used the instructions for Lab 2, this should be named **catsndogsECStasksandservices**.
   3. For **Template file**, enter **template::Lab2-create-ecs-tasks-and-services.yml**

This will use the “template” output artifact from the source step you created earlier, and use the Lab2-create-ecs-tasks-and-services.yml contained within that artifact.

* 1. Leave **Configuration file** blank.
  2. For **Capabilities**, choose **CAPABILITY\_NAMED\_IAM**
  3. For **Role name**, choose the IAM role with **CatsnDogsCloudFormation** in the name, and click **Next Step**.

1. Expand **Advanced,** in **AWS Parameter overrides** enter the following:

**{ "DogTag": { "Fn::GetParam" : [ "MyAppBuild", "build.json", "tag" ] }, "ImageRepo": "<accountid>.dkr.ecr.<region>.amazonaws.com"}**

For example:

**{ "DogTag": { "Fn::GetParam" : [ "MyAppBuild", "build.json", "tag" ] }, "ImageRepo": "123456789011.dkr.ecr.ap-southeast-2.amazonaws.com"}**

**Note:** You can copy the URL of your ECR repository from the ECS Console. Click on Repositories, then on the dogs repository. Copy the URI, but **remove the trailing /dogs**

The parameter override updates the CloudFormation DogTag parameter with the Docker image tag created during the build process. **DogTag** will be replaced with the tag associated with the new image created by the **Build** state, and **ImageRepo** will be replacedwith the URL of your repository.

More information about parameter overrides can be found in the CodePipeline documentation: <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/continuous-delivery-codepipeline-parameter-override-functions.html>

1. In **Input artifacts**:
   1. For **Input artifact #1**,choose “template”.
   2. For **Input artifact #2**,choose “MyAppBuild”.
2. Click **Add action**
3. Click **Save pipeline changes**.
   1. Deploy a new version of the Dogs application

The development team at catsndogs.lol would like you to deploy a new revision of the Dogs application, to test the pipeline. You will do this by uploading a new revision of the Dogs application to the S3 bucket that is being monitored for changes by CodePipeline.

1. Sign in to the AWS management console, click on **Services**.
2. In the **Storage** section, click S3.
3. Click on the S3 bucket with CodeUploadS3Bucket in the name.
4. Click **Upload**.
5. Click **Add files**.
6. From the **Lab-6-Artifacts/v2** folder, upload **dogs.zip.**

This version of dogs includes a newly styled background.

1. On **Set permissions**, click **Next**.
2. On **Set properties**, click **Next.**
3. On Review, click **Upload**.
4. Open the AWS management console, and open the AWS CodePipeline console at <https://console.aws.amazon.com/codepipeline/>.
5. To verify your pipeline ran successfully:
   1. From the **All Pipeline** table, click the **CatsnDogsPipeline,** to monitor the progress of your pipeline.
   2. The status of each stage should change from **No executions yet** to **In progress**, and then **Succeeded** or **Failed**. The pipeline should complete the first run within a few minutes.
6. Copy the value of the **LoadBalancerDNSName**, created by the **catsndogssetup** CloudFormation stack that was deployed at the start of the workshop, in to you address bar of your web browser.

The Dogs application page should appear with fancy new background color.

The build process for the **dogs** Docker image uses the AWS CLI to copy the latest dog memes from an S3 bucket. Although the images are publicly readable, any S3 operation requires AWS credentials. In this case, the credentials from the build environment need to be passed through to the Docker build process, otherwise the build process will fail with “**Unable to locate credentials**”. More details can be found here: <http://docs.aws.amazon.com/codebuild/latest/userguide/troubleshooting.html#troubleshooting-versions>

**Extension activity:** Examine the buildspec.yml file in the dogs.zip file, to understand the steps the CodeBuild project is taking to build and push the docker image. How is the image tagged? How does the CodePipeline pipeline retrieve the tag, to use as a parameter when updating the CloudFormation stack?

Lab 7

Advanced Deployment Techniques

Overview

Now you have a working automated deployment pipeline. Management are extremely happy. However, some buggy code which made its way in to the most recent release of cats, took the cats service offline for a while. **The cat lovers were not happy**.

To address this problem, management have asked you to come up with a safer way to deploy updates, an approach that allows an easy roll back to previous versions, in the event of a problem.

You will setup a blue-green deployment solution, which, because we love cats so much, incorporates some canaries. This solution will allow you to release new versions of the cats application in a staged approach, whilst maintaining a running copy of the previous version for quick roll-back.

The blue-green deployment method will use CloudWatch Events to detect new containers being created. If those containers are part of the a “green” deployment, the CloudWatch Event will trigger a Lambda function. The Lambda function will invoke a Step Functions state machine which performs health checks and gradually moves traffic to the new deployment. The state machine will perform health checks, failing back to the existing stack in the event of a health check failure.

More information about this can be found on the awslabs github repo:

<https://github.com/awslabs/ecs-canary-blue-green-deployment>

* 1. Deploy the lab prerequisites

This step will use CloudFormation to create prerequisite resources which include:

* A new set of the **cat**, **dog** and **simplehomepage** tasks and services that will be used as a deployment target for future application updates. These are prefixed with “green-“.
* A second Application Load Balancer to serve the new green services.
* A set of Lambda functions and an AWS Step Functions state machine.
* A Route 53 hosted zone with the www.catsndogs.lol record set.

1. In the AWS console ensure you have the correct region selected. The instructor will tell you which region to use.
2. In the **Management Tools** section, click **CloudFormation**.
3. Click **Create Stack**.
4. Select **Upload a template to Amazon S3**, then click **Choose File** and choose the file named **Lab7-create-ecs-green-tasks-and-services.yml.**
5. In the Stack name, enter **catsndogsECStasksandservices-green.**
6. Leave the **ECSCluster** and **LabSetupStackName** parameters at their default, unless you changed the name of the CloudFormation stack from the Lab setup, or named the ECS cluster something other than **catsndogsECScluster**.
7. Click **Next**, then click **Next** again.
8. Tick the **I acknowledge that AWS CloudFormation might create IAM resources with custom names** check box.
9. Click **Create.**
10. Wait until the stack status is **CREATE\_COMPLETE.**
    1. Check DNS

The CloudFormation template created a Route 53 hosted zone www.catsndogs.lol in your account. This zone is not registered by the DNS registrar, so it is only accessible if you directly query the zone’s nameservers.

1. In the AWS console ensure you have the correct region selected. The instructor will tell you which region to use.
2. In the **Networking and Content Delivery** section, click **Route 53**.
3. Click the catsndogs.lol hosted zone.
4. There are two record sets for www.catsndogs.lol. One is an ALIAS for the Application Load Balancer with catsn-catsn in the name and has a weight of 100. The other is an ALIAS for the Application Load Balancer with Lab7 in the name and has a weight of 0.
5. Click the catsndogs.lol NS record set and copy one of the values, for example: ns-1478.awsdns-56.org.
   1. If you are using OSX, open the Terminal and type:

dig www.catsndogs.lol @<nameserver\_value>

For example:

dig www.catsndogs.lol @ns-1478.awsdns-56.org.

* 1. If you are using Windows open a command prompt and type:

nslookup www.catsndogs.lol <nameserver\_value>

For example:

nslookup www.catsndogs.lol ns-1478.awsdns-56.org.

1. You should see an IP address in the response section. This is one of the addresses of the Application Load Balancer with catsn-catsn in the name.
2. Open this IP address in your browser. You will see the catsndogs homepage. Click on the **I love cats** and you should see version 2 of the cats page from previous labs. This mimics the DNS lookup process real systems would use if the zone was registered.
   1. Create a Step Functions state machine

In this step, you will create a Step Function state machine which will update the weight associated with the www.catsndogs.lol weighted record set, and perform health checks to ensure the green service is responding. If the step function detects a failure of the green service, it will automatically fail-back to the original configuration.

1. Sign-in to the AWS management console and open the **AWS Step Functions** console at <https://console.aws.amazon.com/states/>
2. If you see the introductory page, choose **Get started**, otherwise, click **Create a state machine**.
3. For **Step 1: Name your state machine**, enter a name for your state machine. Record this information because it will be needed in later steps.
4. For **Step 2: Select a blueprint**, choose Custom.
5. For **Step 3: Review your code and visual workflow**, paste the contents of the **Lab-7-Artifacts/step-functions.json** file, in to the editor window.
6. For each of the **change\_** steps, update the **Resource** key with the name of the Lambda function being used for Route53 record updates. This function will have **CatsnDogsupdateRoute53** in the name.
7. For each of the **check\_** steps, update the **Resource** key with the name of the Lambda function being used for carrying out the health check. This function will have **CatsnDogscheckHealth** in the name.
8. Click the  icon on the Visual Workflow window, to visualize the state machine.
9. Click **Create State Machine**.
10. Choose the IAM role with **StatesExecutionRole** in the name, click OK.
11. At this point **DO NOT** click **New execution**. Instead, click **Dashboard**.
12. Copy the **ARN** of the state machine that you have just created to the clipboard.

**Note:** You now need to add the state machine ARN as an environment variable to the Lambda function, so the Lambda function invokes the correct state machine:

1. Click on the **Services** drop down menu and in the **Compute** section click **Lambda**.
2. Locate the Lambda function with **CatsnDogsHandleECSEvents** in the name and click on it.
3. Expand the **Environment Variables** section.
4. Update the **STEP\_FUNCTION** environment variable with the ARN of the step function state machine.
5. Click **Save**.

The Step Function state machine uses DynamoDB to maintain state, and to link your original service with its newer green service and related information.

This is necessary because Amazon ECS Events can send events on an "at least once" basis; this means you may receive more than a single copy of a given event. Additionally, events may not be delivered to your event listeners in the order in which the events occurred. You will use a DynamoDB table to keep track of state, so the Step Function does not trigger the process more than once.

* 1. Configure CloudWatch Events

In this step, you will create and configure a CloudWatch Event rule that will be triggered when there is a change in task or container state.

1. In the AWS Console, ensure you have the correct region selected.
2. In the **Management Tools** section click **CloudWatch**.
3. Click **Rules**.
4. Click **Create rule**.
5. Choose **Event pattern.**
6. In **Build event pattern to match events by service**:
   1. For **Service Name** choose **EC2 Container Service (ECS).**
   2. For **Event Type**, choose **State Change**
   3. For **Specific details type(s)** choose **ECS Task State Change**.

**Note:** This event pattern triggers the CloudWatch Event rule when and event of type **ECS Task State Change occurs**.

1. From Targets, click **Add target\***
2. Select the Lambda function with **handleECSEvents** in the name.

The CloudWatch event will trigger on all container status changes within the cluster, and will invoke the Lambda function. The Lambda function will be filter the events, acting only on those events that relate to starting a new green version of the cats application.

Because you are configuring this rule using the management console, CloudWatch Events will add the necessary permissions for the Lambda function, so that it can be invoked when the rule is triggered.

If you are creating the CloudWatch Event rule using CloudFormation, the AWS CLI or using one of the AWS SDKs then the policies and roles will need to be created. More details can be found here:

<http://docs.aws.amazon.com/AmazonCloudWatch/latest/events/auth-and-access-control-cwe.html>

1. Click **Configure details**.
2. In Name, enter **catsndogECSRule.**
3. In **Description**, enter “**A rule that is triggered when a new ECS task starts**”.
4. Click **Create**.
   1. Test the deployment

In this task you will mimic the deployment process a Continuous Deployment pipeline would use. You are doing this manually to control the timing.

1. Sign-in to the AWS management console and open the EC2 Container Service (ECS) console at <https://console.aws.amazon.com/ecs/>
2. In the ECS console click **catsndogsECScluster** and then click the **Tasks** tab.
3. Click the service using the **cats-green** task definition. Click **Update**.
4. In **Number of tasks** enter **3**.
5. Click **Next step**, then click **Next step**, then click **Update service.**

This start new tasks using the cats-green task definition. The CloudWatch Events rule you created in earlier will trigger when the new the cats-green container starts. The Lambda function with **handleECSEvents** in the name will be invoked, which then invokes the Step Functions state machine to gradually update the Route 53 weighted record sets.

1. In the AWS console, click on the **Services** menu and in the **Application Services** section click **Step Functions.**
2. Click on the name of the state machine that you created.
3. You should see an execution of the state machine listed, with a state of **running**. Click on the state machine execution to view more details.
4. Observe the **Visual Workflow**. The Step Function state machine performs the following steps:
   1. The Step Function calls the **CatsnDogsupdateRoute53** Lambda function, which updates the Route 53 record set for the green service to have a weight of **10**, and the record set of blue to have a weight of **90**.
   2. The Step Function pauses for **60** seconds.
   3. The Step Function calls the **CatsnDogscheckHealth** Lambda function, which queries the health of the targets registered to the target group associate with the target group attached to the green service.
   4. At this point, if the **CatsnDogscheckHealth** Lambda function returns a response of “healthy” to the Step Function, the Step Function continues to the next step. If the **CatsnDogscheckHealth** Lambda does not return a “healthy” response, the Step Function will move to the fall back step. The fall back step calls the **CatsnDogsupdateRoute53** which updates the blue service to a weight of 100, and the green service to a weight of 0.
   5. The Step Function repeats the **change**, **check**, **decide** steps for 2 more cycles, until the green record set has a weight of **100** and the blue record set has a weighting of **0**. At which point, all traffic will be routing to the green service.
5. Now the Route 53 record set has switched over to the new Application Load Balancer, look up the IP address again:
   1. If you are using OSX, open the Terminal and type:

dig www.catsndogs.lol @<nameserver\_value>

For example: dig www.catsndogs.lol @ns-1478.awsdns-56.org.

* 1. If you are using Windows open a command prompt and type:

nslookup www.catsndogs.lol <nameserver\_value>

For example: nslookup www.catsndogs.lol ns-1478.awsdns-56.org.

1. You should see an IP address in the response section. This is one of the addresses of the Application Load Balancer with Lab7 in the name. This again mimics the behavior real systems would use if the zone was registered.
2. Open this IP address in your browser. You will see the catsndogs homepage. Click on the **I love cats** and you should see version3 of the cats page.
   1. Extension Exercise

It has come to the attention of the bean-counters that that the CloudWatch Event rule, **catsndogECSRule**, is being triggered more often than the number of times a new service deployment occurs. This is bad because unused events were swapped out for a steady supply of catnip.

For the sake of the cats, please investigate why this is happening, and implement a solution that reduces the number of times the **catsndogECSRule** triggers.

1. In the AWS console ensure you have the correct region selected. The instructor will tell you which region to use
2. Click on the **Services** drop down menu and in the **Compute** section click **Lambda.**
3. Locate the Lambda function with **CatsnDogsHandleECSEvents** in the name and click on it.
4. Click on **Monitoring**, to view the CloudWatch metrics for the Lambda function.
5. Click **View logs in CloudWatch**, to see the execution logs for the Lambda function.
6. Locate the most recent Log Stream and click on it.
7. Explore the log to see if anything erroneous stands out. What things could be done to resolve the issue?

Clean up

Overview

**Congratulations!** You have successfully helped the team at catsndogs.lol build a highly scalable container based application architecture and an automated deployment pipeline.

This step cleans up all the resources you have created in previous labs.

Clean up instructions

1. Delete the ECS Cluster. This will also delete all the ECS Tasks and Services within the cluster.
2. If it exists, delete the Auto Scaling Group **ECS-On-Demand-Group**
3. If it exists, delete the Launch Configuration **On-Demand-ECS**
4. Verify the cats, dogs, simplehomepage, MXnet, and ImageAssessor tasks are deleted.
5. Delete the cats, dogs and simplehomepage ECR repositories if they exist.
6. Delete the Parameter Store secure string named **UnicornLocation**.
7. Delete the CloudWatch alarms ScaleDownSpotFleet and ScaleUpSpotFleet
8. Delete the CloudWatch events ImageAssessor and HandleECSEvents.
9. Delete the CloudWatch Logs log groups for:
   1. aws/codebuild/dogs-build
   2. All log groups beginning with aws/lambda/Lab7
10. Delete the CodePipeline pipeline.
11. Delete the CodeBuild project.

**[continued below]**

1. Empty and delete the CodeUploads S3 bucket.
2. Delete the Step Functions state machine.
3. Delete the Route 53 A-type record sets inside the catsndogs.lol hosted zone.
4. Delete the CloudFormation stacks you created. Because later labs rely on the stacks from earlier labs, you should delete the Lab0 stack only after the others have reached the DELETE\_COMPLETE state:
   1. Lab7: **catsndogsECStasksandservices-green**
   2. Lab2: **Lab2-create-ecs-tasks-and-services** and **Lab2-loadgenerator**
   3. Lab1: **Lab1-add-ondemand-asg-to-cluster**
   4. Lab0: **catsndogssetup**

“Advanced Container Management at catsndogs.lol”

Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.