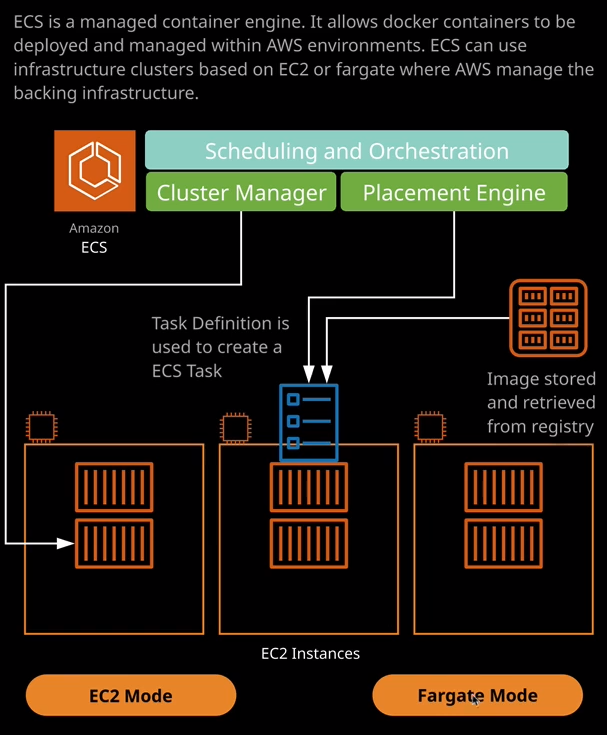
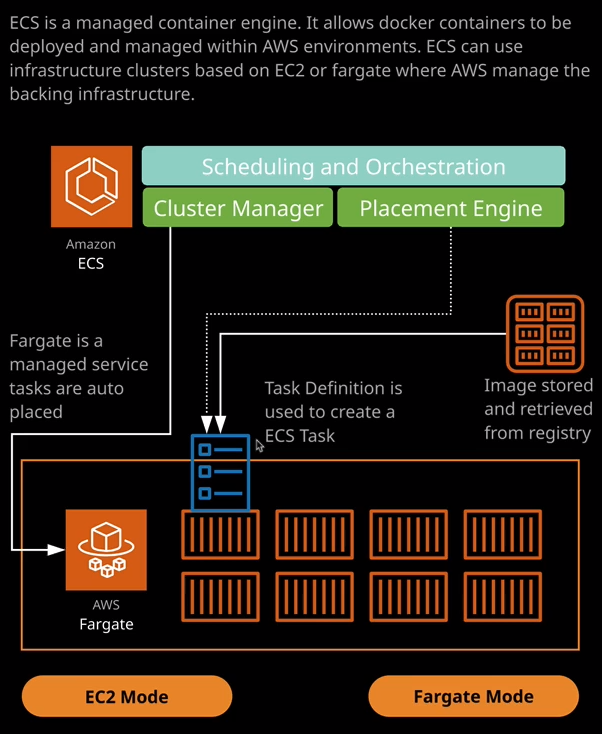
**ECS or the elastic container service**

**ECS is a managed service inside AWS that allows you to host containers. It allows you to run data containers quickly and easily inside AWS with less management overhead than using EC2, especially at scale.**

In the last lesson, you learned about Docker and how you can use Docker which is a container engine running on top of EC2 to handle simple Docker deployments. Well **ECS extends that and allows you to manage containers using infrastructure that AWS either partially manages or fully manages**.



Now ECS **runs in two different modes. It runs in EC2 mode, which allows you to utilize EC2 instances, but those that provisioned by ECS and those EC2 instances are managed by ECS, which includes a scheduling and orchestration component, a custom manager component, and a placement engine component**. Essentially, you can use a **registry**. So an example of a registry is **Docker Hub** where you uploaded the container cat image that I created in the previous lesson. Well that's used to pull images down and deploy those onto ECS hosts so these are EC2 hosts that are running all of the ECS agents and let you deploy containers.

  
Now there is also what's known as **Fargate** mode. **So it uses container hosts that are completely managed by AWS.** So you've **still got the same core components of ECS which includes scheduling and orchestration cluster management and the placement engine. Now with this, you have no visibility of EC2 instances. It's all managed as a service.** You simply direct it about the type of containers that you want to use, where you want them deployed, how you want them deployed, and it manages the process end to end. You don't have to worry about EC2 instances, and you don't have to worry about managing them.

So let's have a look at how this works. So I'm going to go back to the AWS console. Now I want to step through the different entities involved in ECS and it's probably easier to do this with a quick demo, so I'm going to click on Clusters and then Create a Cluster**. A cluster inside ECS is essentially the container it's a container of configuration that defines how you want a specific grouping of ECS infrastructure to work, and you can deploy a cluster in one of three modes:**Two of those use EC2 instances. So using this EC2 mode, that's on the right of the screen. **Those are called EC2 Windows plus networking and EC2 Linux plus networking.** So you can decide to use ECS using Windows as the base operating system or Linux but in both of those cases, you do have EC2 instances in your account that you do need to be aware of.   
**The third option**, though, is to utilize **Fargate**, which is this serverless approach. It's managed entirely by AWS, and that's why I'm going to use for this demonstration.

So I click on next step, you'll need to name the cluster, so I'm going to call it demo. Remember, this is going to be a Fargate mode cluster, so I won't need to worry about any EC2 instances, so I'll call it demo. You can choose to create **a Fargate cluster inside a custom VPC**. You remember we haven't covered VPCs yet. That is in section four of the course. For now, I'll leave this on selected, which will use the default the VPC and I'll click on Create, and that's all there is to it that's created a Fargate cluster. So that's the grouping of infrastructure that AWS will use to deploy our containers to.

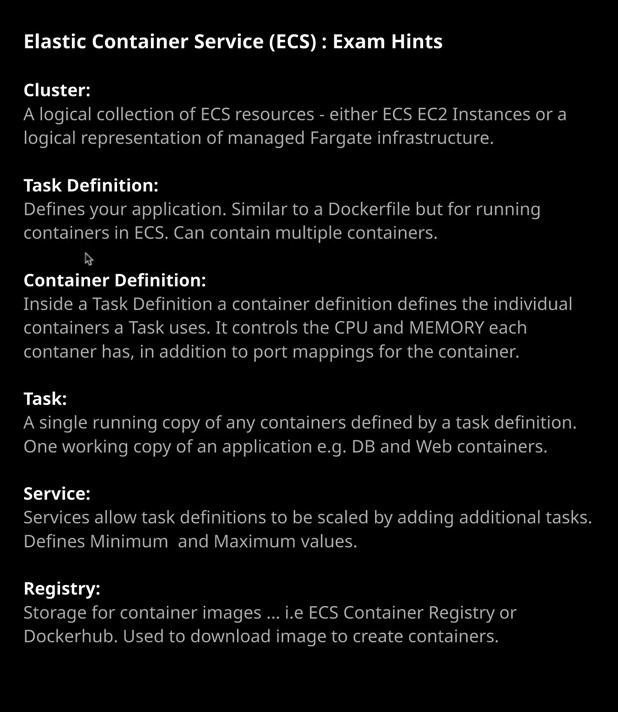
Now, **the first thing that we need to do when we create an ECS cluster is to create what's known as a task definition**. So remember how we have the Docker file in the previous lesson and a Docker file was essentially a definition of how to create an image. Well, **the task definition is essentially the definition of what's going to run inside ECS**. **So a task definition tells ECS how to run your containerized application, and that's needed because an application might itself consists of a number of containers, so one application might be multiple containers, and so a task definition contains all of the information that's needed end to end for your application.**   
So I'm going to create a task definition. I'll go to task definitions, create a new definition, I'll need to select that it's Fargate launch type because I'll be putting it into a Fargate ECS cluster. You had the option of picking EC2 based or Fargate. I'm using Fargate as a demo. So I'm going to select that and click on next step. I'm going to name the task definition containercat.

You also have the option of creating a **task role**. Now, **a task is a copy of your application running. So this task definition could be used to create one or more running copies of a given application and a task role is how we can give the application the permissions to interact with other AWS resources which just like an EC2 instance role it's just like a Lambda execution role and in this case, it gives those permissions really granular and really focused on a specific task.** It's important to know for the exam if **you need to give an individual container or a group of containers access to permissions, then you use a task role.**

Now you're also able to **specify the amount of resources to give to a particular task. So these are the resources that will be given to the task as a whole, you're able to subdivide those per container**, but for this example, we're going to utilize one gig of memory in naught 0.5 virtual CPU. So that's the resources that this task is going to have and because we're only using a single container it will be the resources the whole container gets.

Now I mentioned **that task definition can define multiple containers**. In this case, we're going to define one as part of this task definition but it's here where you'll define the individual containers. So I'm going to click on Add Container. I'm going to name this container cat container web because it's running a web server, and it's part of the cat container definition. So I'll keep the naming scheme consistent and then under image I need to specify where the image for this container is located, and it's on Docker Hub. So I'll need to specify it in this format. So it's docker.io and then my username, which is acantrill and then containercat, which is the name of the image. So, docker io /a.cantrill/containercat and then for memory, I'm going to put 1024 which is going to allocate 1024 MB to this particular container and then for port mapping I need to port 80.   
Remember how I mentioned in the previous lesson **how containers themselves map port from the container to the outside world.** So it's port 80. That's what you used on the container, and that's what I want mapped to the outside world. So I'm going to type port 80 here for the port and then I'll leave everything else as default and click on Add. So what I've just done there is **create the container definition**.   
So how exactly this individual container inside the task definition works**. So this is going to load the image from Docker Hub, which I uploaded** in the previous lesson. **So that's going to be our container cat application, and that's the only container definition that's running inside this task definition**. I'll go ahead and click on Create. So that's the task definition but by itself, it doesn't do anything. Essentially what you need to do to make this accessible is **from a task definition you need to create a task and the task is a running copy of a task definition.**   
So to do that I'm going to go to tasks and I'm going to click on run new task. The launch type will need to be Fargate because this is a Fargate cluster that I'm using. The task definition itself will be container cat. The platform version will be latest. I'll need to select the cluster that I'm going to deploy this task to. So I only have one in this account. So it's demo. I'm only going to be running one copy of this task so I can leave that at one and then I'll **need to specify the VPC and security groups that this container will use**. So I'm going to go ahead and select the VPC that's used by this cluster and then I'm going to go ahead and pick each of the subnets inside this VPC.

This container should have the option of using all of the subnets inside this VPC. Now it does have access to a security group. If I click on edit, you'll see that it's allowing port 80 and that's because I added that port 80 mapping when I created the task definition, and it's also set to **auto assign a public IP**. So it will be available from the public internet. So at this point I can go ahead and run the task. Now while that's creating, I want to talk about services.



So **services inside ECS fit quite closely with tasks and task definitions**. So I talked at the start this lesson, how we got **a cluster and that's a logical collection of ECS resources. A task definition defines your application. It's similar to a Docker file, but for running containers in ECS and a task definition can have multiple containers defined inside it and that's what a container definition is. The container definition defines the resources your container has and what ports it has**.  
Now you take a task definition and you can create a single task, which is what I just did here but what if you want to auto scale? What if you want to be able to cope with increasing or decreasing load of people visiting containercat? Well, that's where services come in. **A service allows you to create multiple instances of that task for a given task definition, so a task definition might have a web server and a database server, and that represents one copy of that application but if your load increases, you might want to define minimum and maximums so that you can auto scale and that's what a service does**. You can create a service. You can specify various criteria. The minimum and maximum levels that you want that task to scale up and down to and it can automatically create additional tasks for a given task definition. So **that's how you can scale a containerized application up or down by using services.**

So **a service is how you take a task definition and run multiple tasks automatically from that definition.** What we've done in this example is create this single task. To demonstrate how this works, I'm going to open up this task. It'll have a public IP for this task. Remember, this is a container that's automatically running inside ECS using Fargate mode running on infrastructure that you don't have to manage if I go ahead and open the IP in a separate tab then we've got another cat in a container in a container. So this is a copy of the application that's running inside our ECS cluster using the Fargate cluster mode. So it requires no admin overhead to make this work. So that's pretty much what I wanted to cover in this lesson. I just wanted to introduce you to the two different modes, **so EC2 mode where you can utilize Linux or Windows operating systems running on EC2 instances that you deploy and manage within your cluster and then Fargate mode where AWS handles it completely end to end, and you get no exposure to any infrastructure but essentially you utilize ECS when you do have a containerized workload, maybe a used to using Docker, and you want to deploy them and use them inside AWS without having the overhead of managing your own EC2 instances.**