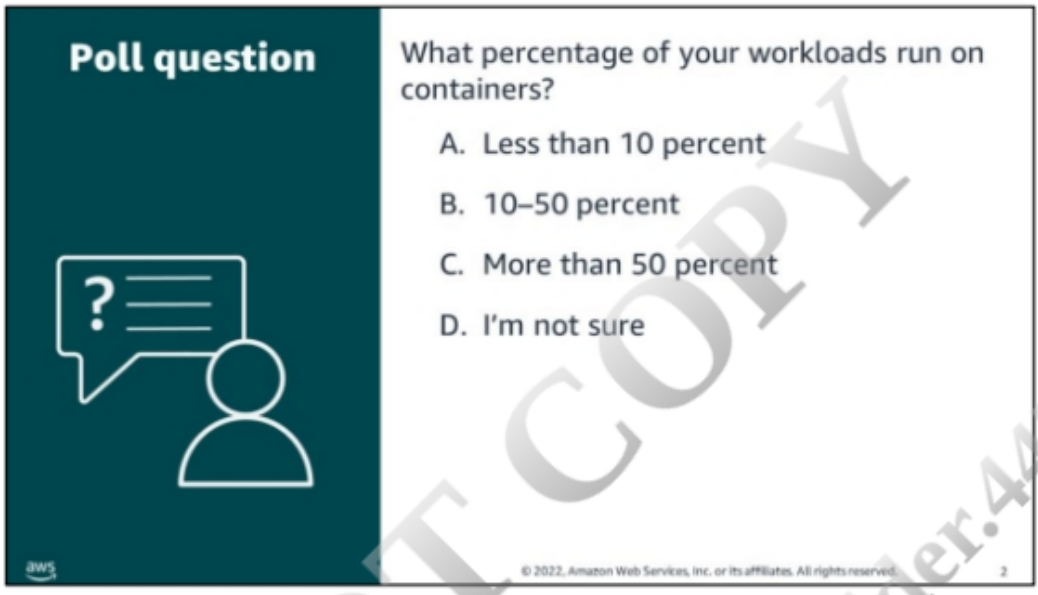
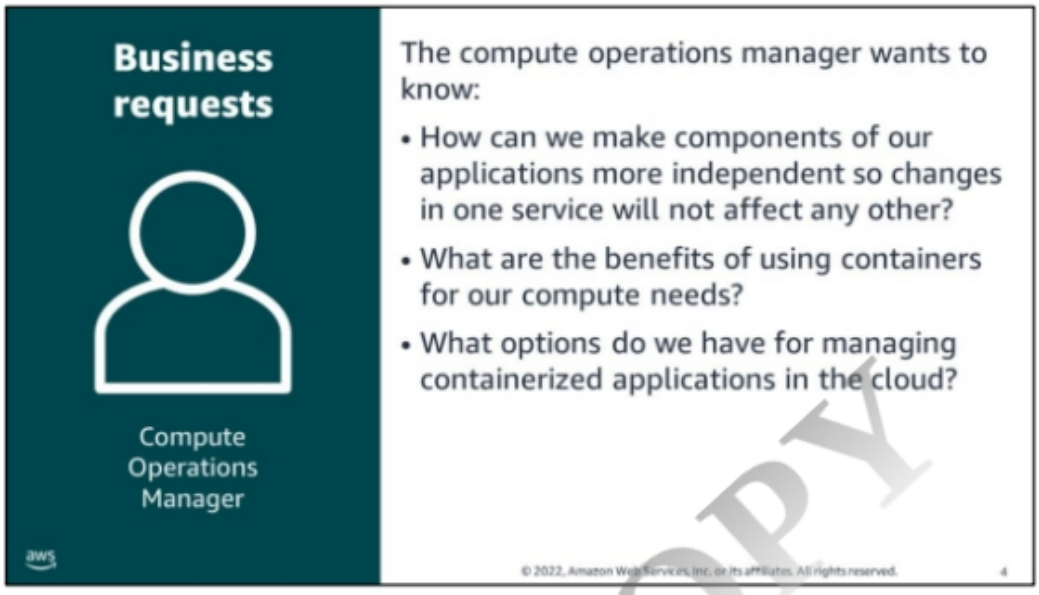
Printed by: 2d35e8483186bd2@placeholder.44518.edu. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Module overview

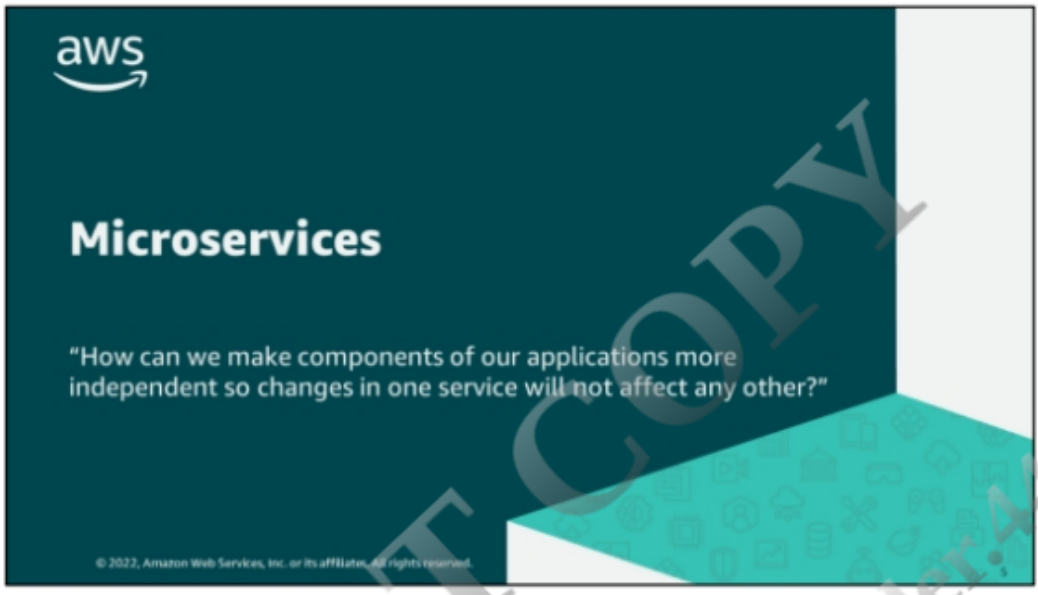
- Business requests
- Microservices
- Containers
- Container services
- Present solutions
- Knowledge check

 3



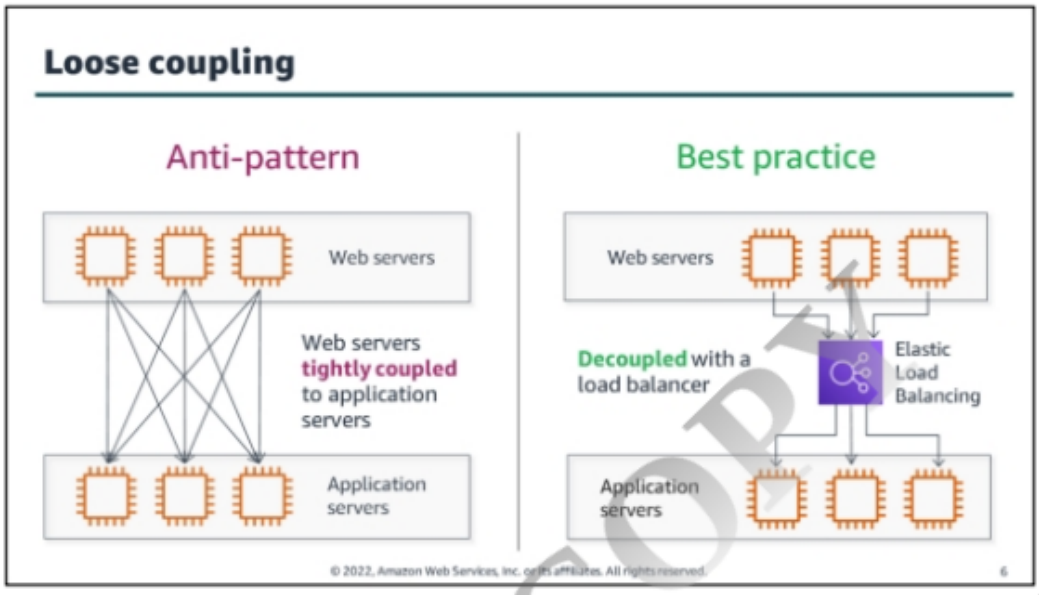
Imagine your compute operations manager meets with you to discuss different application architectures and how you support them in the cloud. Here are some questions they are asking.

At the end of this module, you meet with the compute operations manager and present some solutions.



The compute operations manager asks, "How can we make components of our applications more independent so changes in one service will not affect any other?"

The company has some legacy applications that they want to rearchitect in the cloud. These applications have many interdependent components, which makes them difficult to update and to isolate errors. The company is asking you to research different ways to architect these applications.

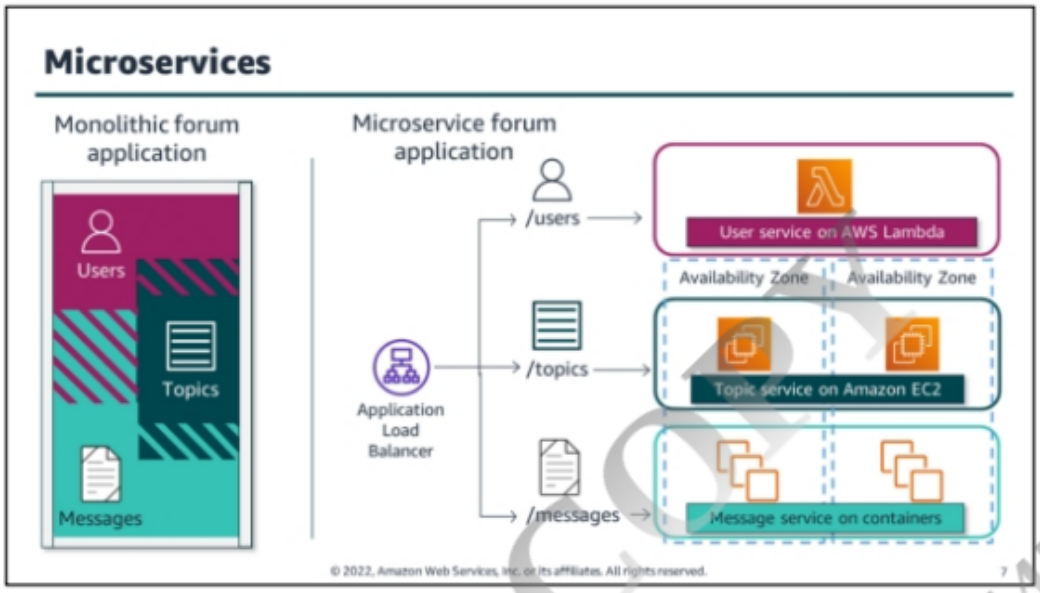


Traditional *monolithic* infrastructures revolve around chains of tightly integrated servers, each with a specific purpose. When one of those components or layers goes down, the disruption to the system can be fatal. This configuration also impedes scaling. If you add or remove servers at one layer, you must also connect every server on each connecting layer.

With *loose coupling*, you use managed solutions as intermediaries between layers of your system. The intermediary automatically handles failures and scaling of a component or a layer. Two primary solutions for decoupling your components are load balancers and message queues.

The diagram on the left illustrates a collection of web and application servers that are tightly coupled. In the tightly coupled architecture, the connection between a web server and an application will give an error if one application server goes down.

The drawing on the right shows a load balancer (in this case, using Elastic Load Balancing) that routes requests between the web servers and the application servers. If a server fails in the loosely coupled architecture, the load balancer will automatically start directing all traffic to the two healthy servers.



Microservices are an architectural and organizational approach to software development. Using a microservices approach, you design software as a collection of small services. Each service is deployed independently and communicates over well-defined APIs. This speeds up your deployment cycles, fosters innovation, and improves both maintainability and scalability of your applications.

Autonomous
The component services in a microservices architecture are isolated from one another and communicate through an API. Because of this, you can develop, update, deploy, operate, and scale a service without affecting the other services. These services can be owned by small autonomous teams, allowing for an agile approach.

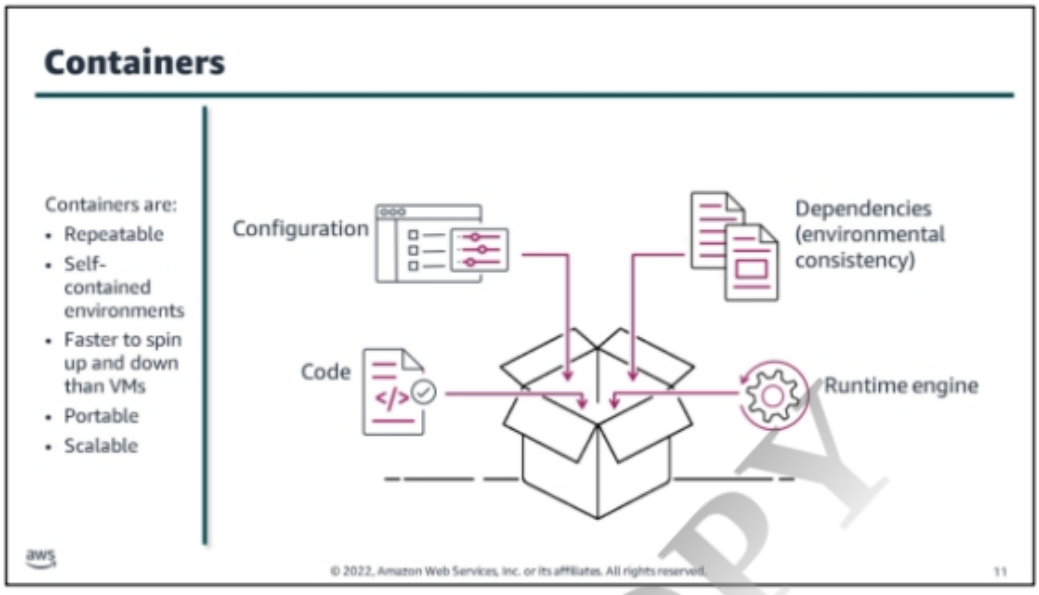
Specialized
You design each service for a set of capabilities that focuses on solving a specific problem. Teams can write each service in the programming languages best suited to that service. They can also host their services on different compute resources.

In this example, a monolithic forum application is refactored to use a microservices architecture: a user service, a topic service, and a message service. The /users service team runs the user service on AWS Lambda. The /topics service team runs the topics service on Amazon Elastic Compute Cloud (Amazon EC2). The /messages service team runs the messages service on containers. The microservices application is distributed across two Availability Zones and manages traffic with an Application Load Balancer.

For more information about microservices architecture, see the AWS Whitepaper, "Microservices architecture on AWS" (https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html).

The compute operations manager asks, "What are the benefits of using containers for our compute needs?"

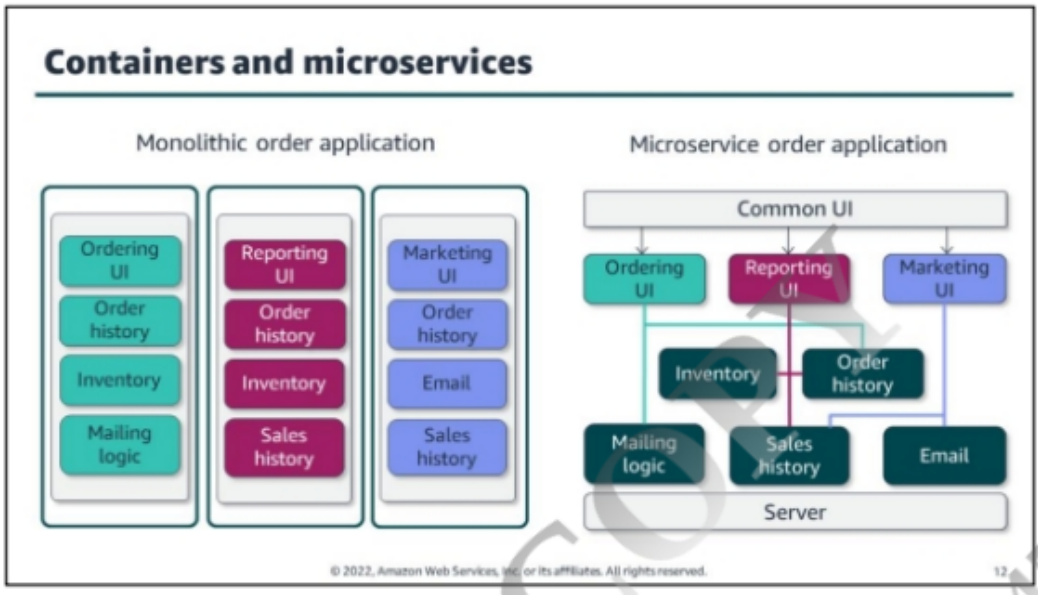
The software engineering team is considering refactoring some of the company's applications to use containers. The compute operations manager is asking you to research the benefits of containers and needs to understand how containerized applications are hosted.



The benefits of a microservice-oriented architecture should trickle down to an infrastructure level. We achieve that with containers.

Although running virtual machines (VMs) in the cloud gives you a dynamic, elastic environment, you can simplify your developers' processes. Containers provide a standard way to package your application's code, configurations, and dependencies into a single object.

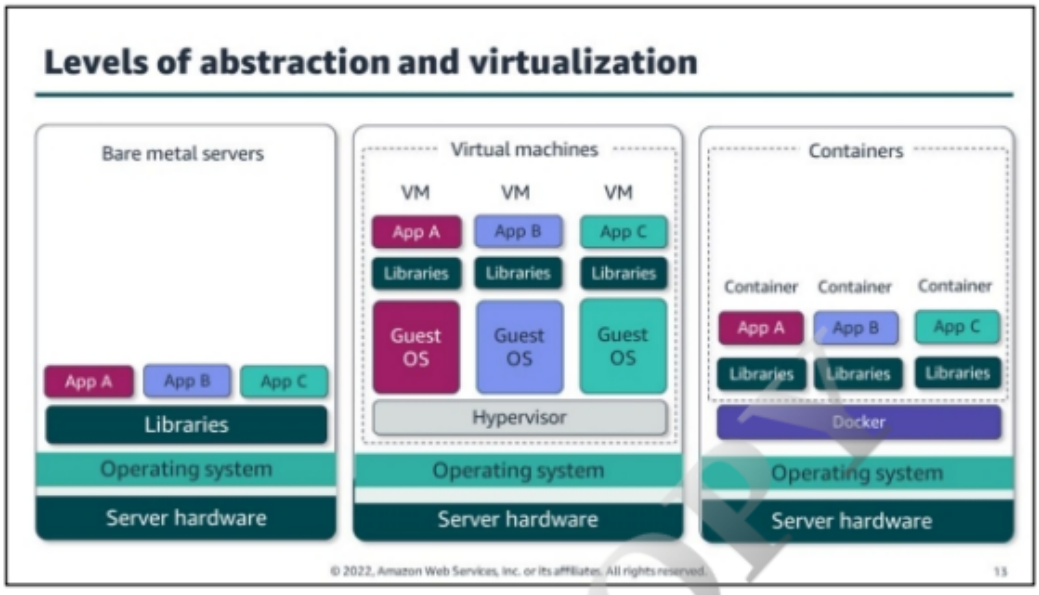
Containers share an operating system installed on the server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of the environment.



Containers are an ideal choice for microservice architectures because they are scalable, portable, and continuously deployable.

Earlier in this module, you learned how microservice architectures decompose traditional, monolithic architectures into independent components that run as services and communicate using lightweight APIs. With these microservice environments, you can iterate quickly, with increased resilience, efficiency, and overall agility.

You can build each microservice on a container. Because each microservice is a separate component, it can tolerate failure better. If a container fails, it can be shut down and a new one can be started quickly for that particular service. If a certain service has a lot of traffic, you can scale out the containers for that microservice. This eliminates the need to deploy additional servers to support the entire application. Microservices and containers are also great for continuous deployment. You can update individual services without impacting any of the other components of your application.



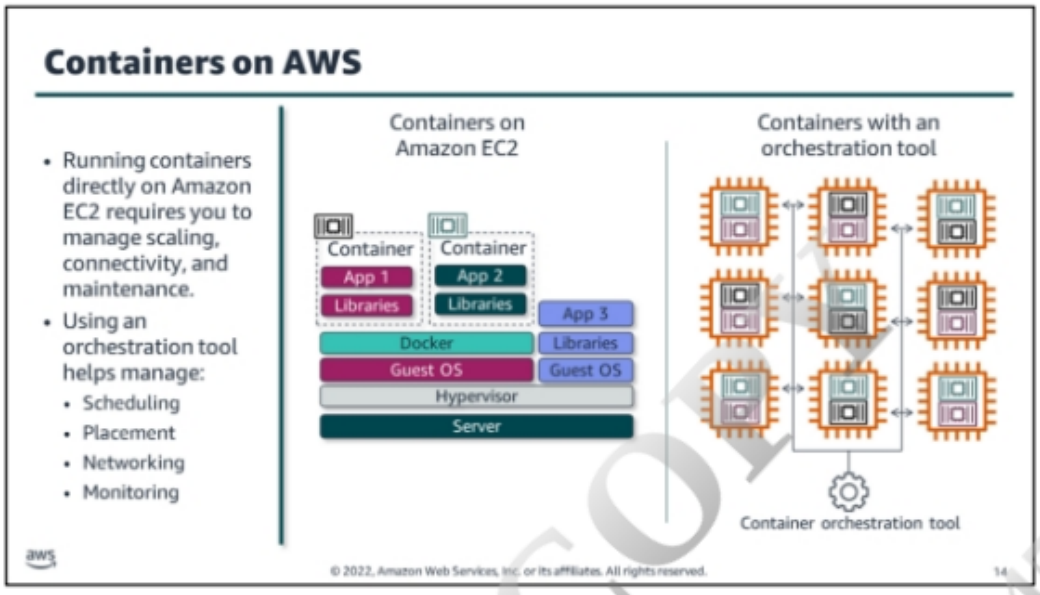
A bare metal server runs a standalone operating system (OS) with one or many applications by using libraries. Costs remain constant, whether the server is running at 0 percent usage or 100 percent usage. To scale, you must buy and configure additional servers. It is also difficult to build applications that work on multiple servers since the OS on those servers would have to be the same. You also need to synchronize the application library versions.

With virtual machines, you isolate applications and their libraries with their own full OS. The downside of VMs is that the virtualization layer is "heavy." Each VM has its own OS. This requires more host CPU and RAM, reducing efficiency and performance. Having an individual OS for each VM also means more patching, more updates, and more space on the physical host.

With containerization, containers share a machine's OS system kernel and the underlying OS file system is exposed. Sharing a machine's OS system kernel allows shared libraries but can permit individual libraries as needed. This makes containers highly portable. You can also start and stop containers faster than VMs. Containers are lightweight, efficient, and fast.

Unlike a VM, containers can run on any Linux system, with appropriate kernel feature support and the Docker daemon. This makes them portable. Your laptop, your VM, your Amazon EC2 instance, and your bare metal server are all potential hosts.

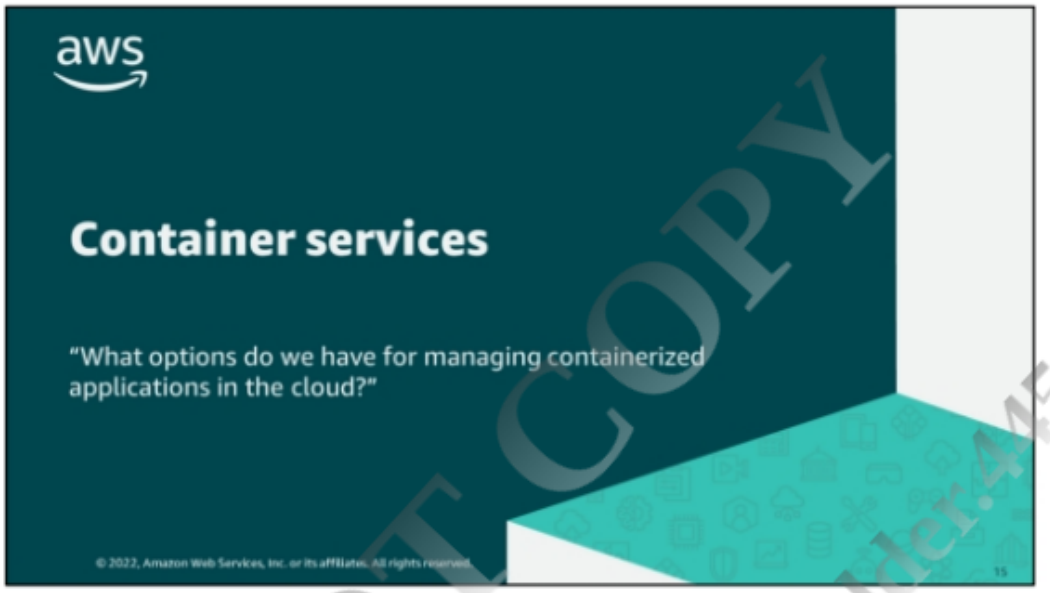
The lack of a hypervisor requirement also results in almost no noticeable performance overhead. The processes are communicating directly to the kernel and are largely unaware of their container silo. Most containers boot in only a couple of seconds.



When running containers on AWS, you have multiple options.

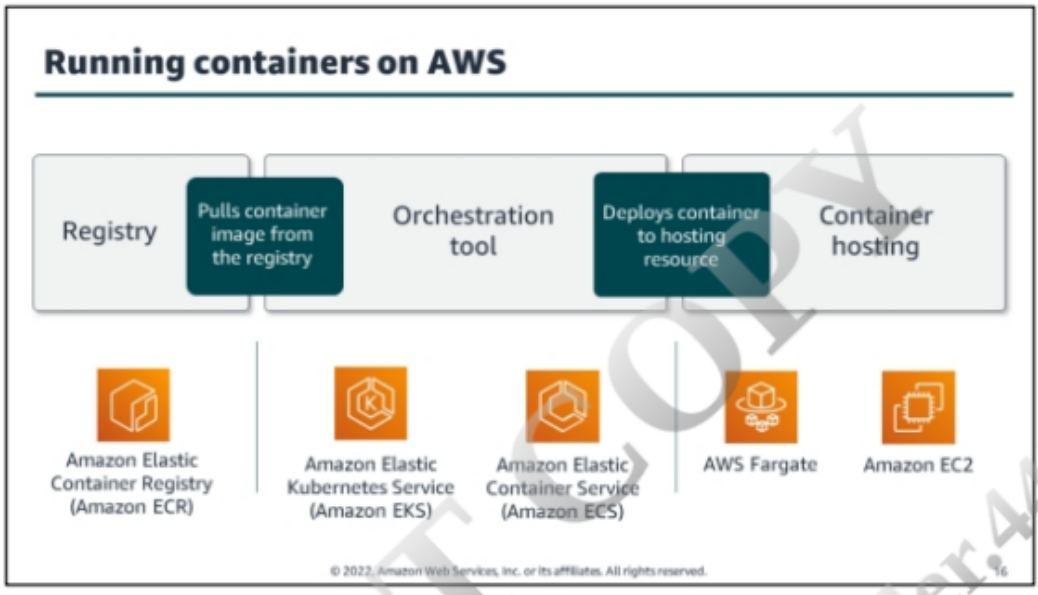
Running containers on top of an EC2 instance is common practice and uses elements of VM deployments and containerization. This diagram shows the underlying server infrastructure—a physical server, the hypervisor, and two virtual guest operating systems. One of these operating systems runs Docker, and the other runs a separate application. The virtual guest OS with Docker installed can build and run containers. Though possible, this type of deployment can only scale to the size of the EC2 instance used. You also have to actively manage the networking, access, and maintenance of your containers.

Using an orchestration tool is a scalable solution for running containers on AWS. An orchestration tool uses a pool of compute resources, which can include hundreds of EC2 instances to host containers. The orchestration tool launches and shuts down containers as demand on your application changes. It manages connectivity to and from your containers. It also helps manage container deployments and updates.



The compute operations manager asks, "What options do we have for managing containerized applications in the cloud?"

The software engineering team is considering refactoring some of the company's applications to use containers. The compute operations manager is asking you to research which AWS services support containerized applications in the cloud.



Deploying your managed container solutions on AWS involves selecting and configuring the following components:

Registry – When you develop containerized applications, you build a container image that holds everything needed to run your container. This includes application code, runtime, system tools, system libraries, and settings. You push your images to a repository for version control and pull those images from the repository to deploy containers. A *registry* is a collection of repositories.

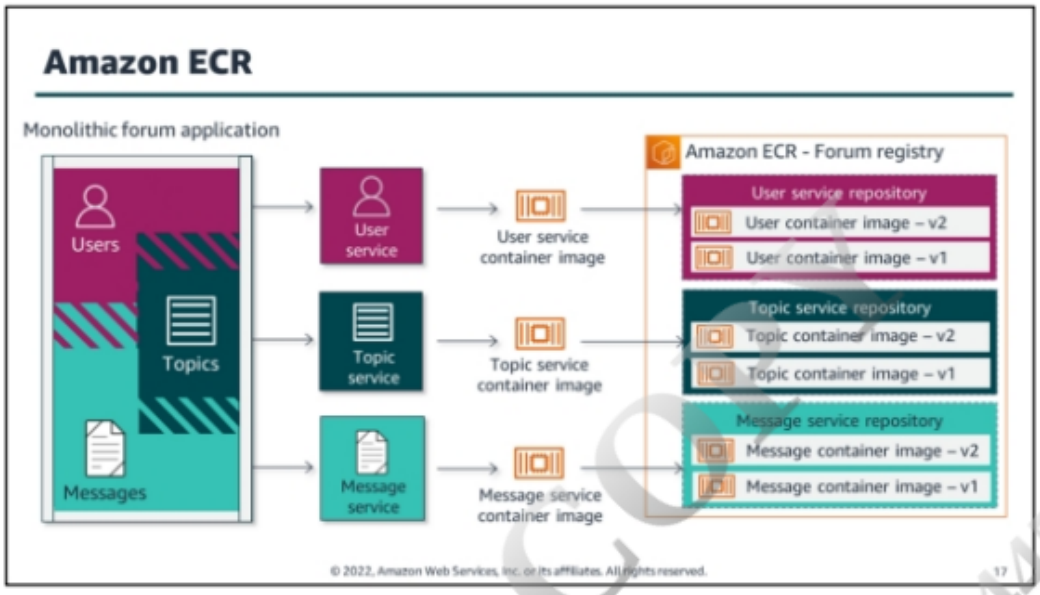
- AWS offers Amazon ECR as a container image registry that supports integration with other AWS services.

Orchestration tool – You use a container orchestration tool to manage your containerized applications at scale. An orchestration tool manages the scaling, networking, and maintenance of your containers. They help you manage containers' startup and shutdowns, monitor container health, deploy updates, and manage failover and recovery.

- Amazon EKS is a managed service that you can use to run the Kubernetes container orchestration software on AWS. You might choose this option if you require additional control over your configurations.
- Amazon ECS is a managed container orchestration service that offers a more managed model for deploying your containers. Amazon ECS features additional integrations with other AWS services.

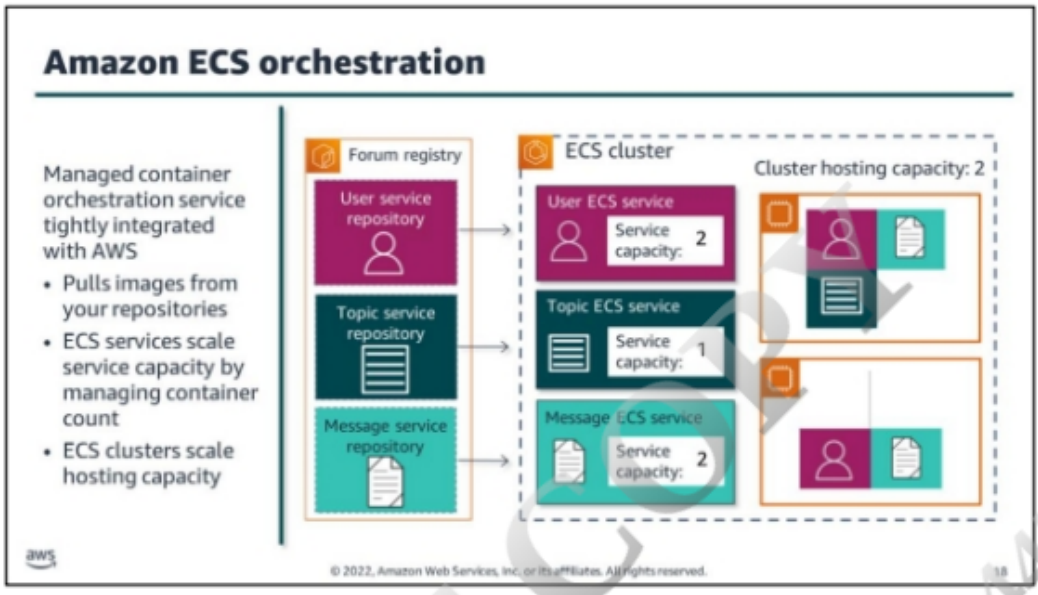
Container hosting – You need to decide which compute resource your orchestration tool will use to host your containers. This is often referred to as the container's *launch type*.

- You can choose Amazon EC2 to launch containers on a variety of instance types. As demand changes, you can scale out and scale in the number of EC2 instances used to host containers. This is a cost effective method that provides more control over the instance type.
- AWS Fargate is a serverless hosting service that automatically allocates CPU and memory resources to support your containers. With Fargate, you do not have to provision or manage the underlying compute.



Amazon Elastic Container Registry (Amazon ECR) is a managed Docker container registry. You push your container images to Amazon ECR and can then pull those images to launch containers. With Amazon ECR, you can compress, encrypt, and control access to your container images. You also manage versioning and and image tags. An Amazon ECR private registry is provided to each AWS account. You can create one or more repositories in your registry and store images in them.

This example refactors a monolithic forum application into microservices using containers. You break apart the code into individual encapsulated services: the user service, the topic service, and the message service. You build container images for each of these services, which can be launched, updated, and shut down independently. In this example, the container image for each service is pushed to its own repository, which stores multiple versions of each image. An orchestration service can pull these container images and deploy new containers as needed.



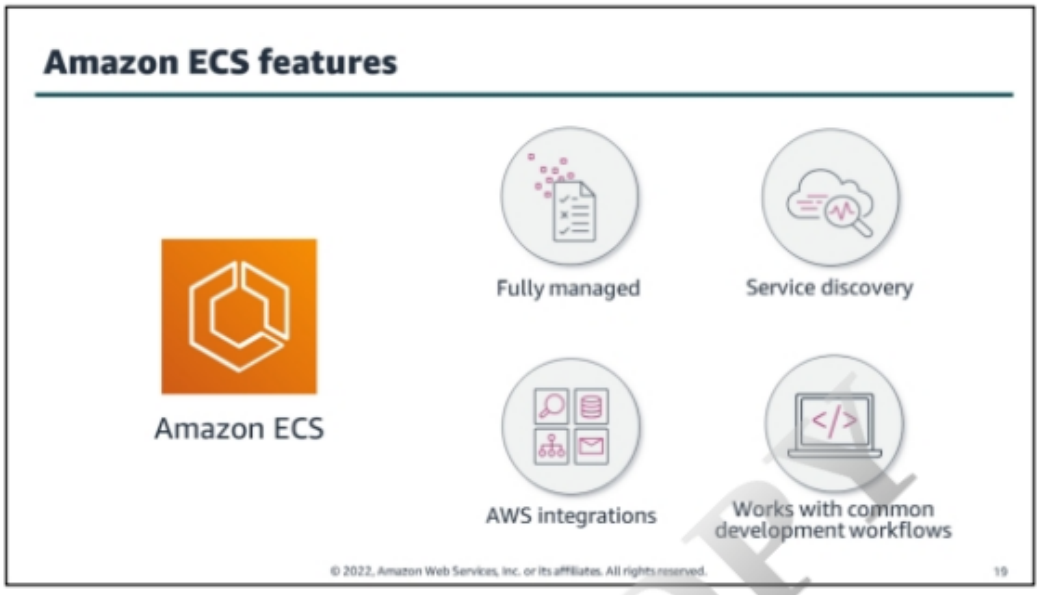
Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance container management service that supports Docker containers. Amazon ECS manages the scaling, maintenance, and connectivity for your containerized applications.

With Amazon ECS, you create ECS services, which launch ECS tasks. Amazon ECS tasks can use one or more container images. Amazon ECS services scale your running task count to meet demand on your application.

You create an Amazon ECS cluster with dedicated infrastructure for your application. You can run your tasks and services on a serverless infrastructure managed by AWS Fargate. If you prefer more control over your infrastructure, manage your tasks and services on a cluster of EC2 instances. Your cluster can scale EC2 hosting capacity by adding or removing EC2 instances from your cluster.

In this example, the containerized forum application is running in Amazon ECS. Three ECS services have been created and added to an ECS cluster: user, topic, and message. The Amazon ECS agent running on the compute service pulls container images for each of these services from the forum registry in Amazon ECR. The cluster is using EC2 hosting with a capacity set to two EC2 instances. The user and message services have a task count set to two, and the topic service has a task count of one.

With Amazon ECS, you don't have to operate your own cluster management and configuration management systems, or worry about scaling your management infrastructure.



Fully managed – As a fully managed service, you don't need to manage the control plane, nodes, or add-ons. This makes it easier for teams to focus on building the applications, not the environment.

Service discovery – Amazon ECS features support for service discovery, which you can use to register your ECS services to Domain Name System (DNS) names. For example, you could register a service called "backend" with a private DNS namespace such as backend.example and a service called "frontend" with frontend.example. You could then configure these services to be able to discover each other within the same virtual private cloud (VPC). With service discovery, your microservice components are automatically discovered and added to namespaces as they're created and shut down.

AWS integrations – Amazon ECS has close integrations with many AWS services, for example, the following:

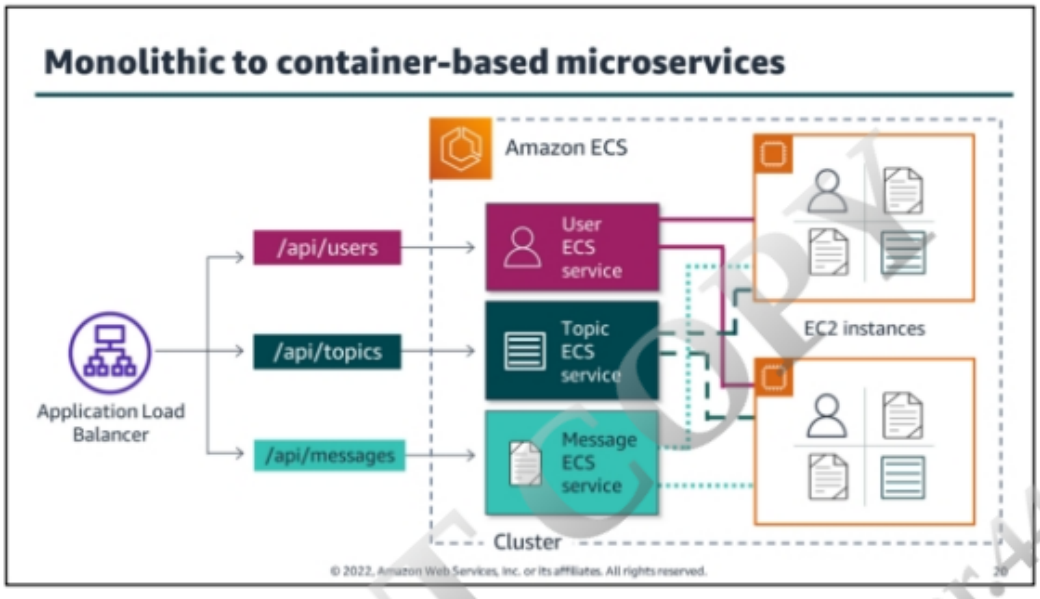
- Amazon ECR: Amazon ECS easily integrates with Amazon ECR, making is easier for your containerized applications to access and run your container images.
- AWS Identity and Access Management (IAM): You can assign granular permissions for each of your containers. This allows for a high level of isolation when building your applications.
- Amazon CloudWatch Logs and Container Insights: You can view the logs from your containerized applications and instances in one convenient location.

Flexible hosting options – With ECS you can use both Amazon EC2 and serverless hosting with AWS Fargate. You can schedule the placement of your containers across your cluster based on your resource needs, isolation policies, and availability requirements.

Development workflows – Amazon ECS supports continuous integration and continuous deployment (CI/CD). This is a common process for microservice architectures that are based on Docker containers. You can create a CI/CD pipeline that takes the following actions:

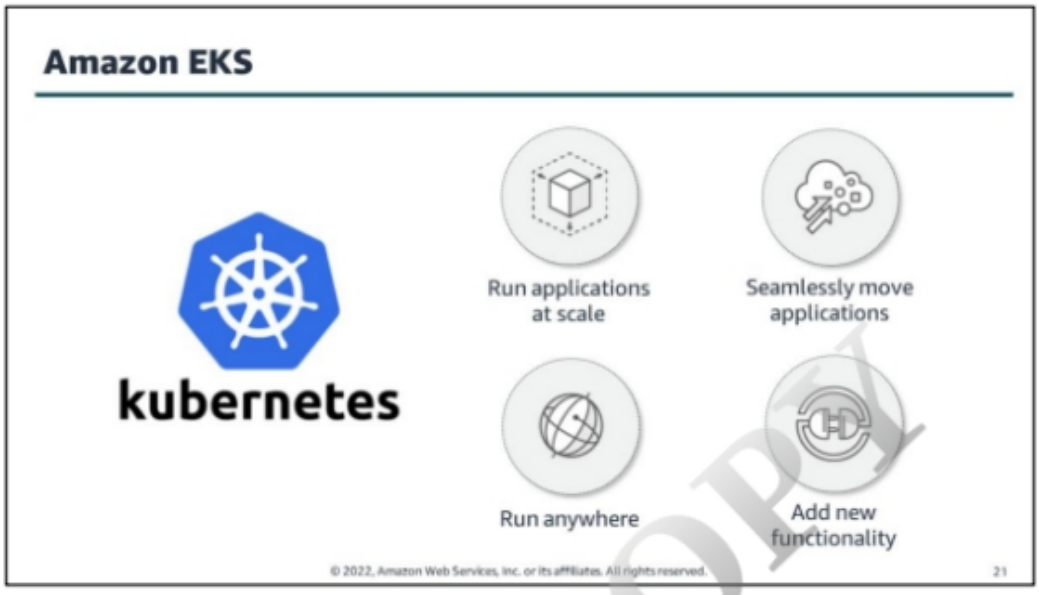
- Monitors changes to a source code repository
- Builds a new Docker image from that source
- Pushes the image to an image repository such as Amazon ECR or Docker Hub
- Updates your Amazon ECS services to use the new image in your application

For more information about service discovery, see "Service Discovery" in the *Amazon Elastic Container Service Developer Guide* (https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-discovery.html).



Earlier in this module, you learned the difference between traditional monolithic infrastructures and microservices. Now you can run the microservices on a managed Amazon ECS cluster.

This diagram shows an application load balancer sending web traffic based on the path of APIs in the request for each service. You register the user service, topic service, and message service with different target groups. When Amazon ECS starts a task for your service, it registers the container and port combination with the service's target group. The Application Load Balancer routes traffic to and from that container.

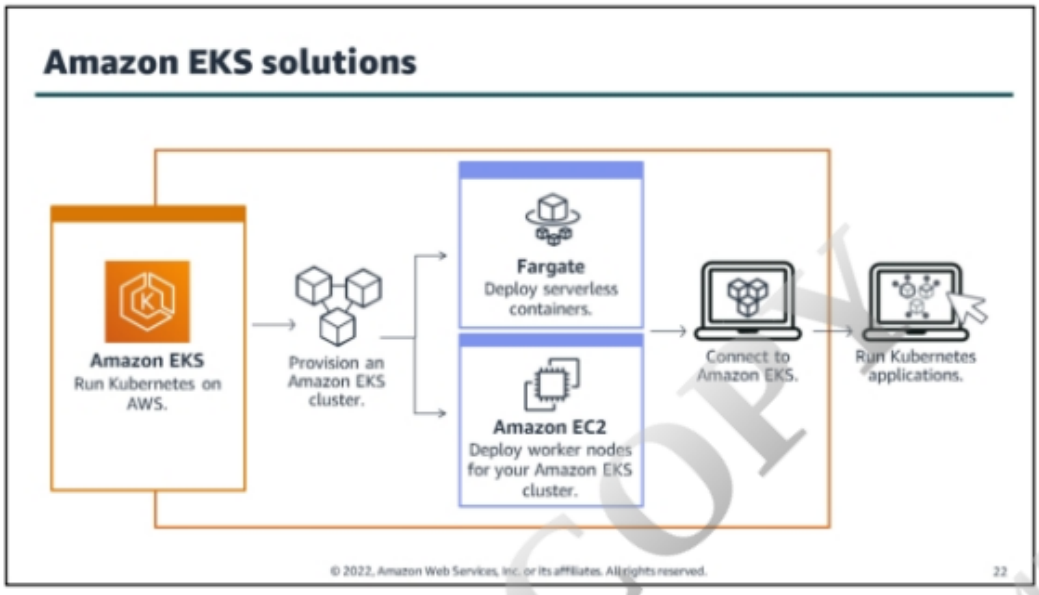


Kubernetes is an open-source software that you can use to deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs containers on those instances with processes for deployment, maintenance, and scaling. With Kubernetes, you can run any type of containerized applications using the same tool set on premises and in the cloud.

Amazon Elastic Kubernetes Service (Amazon EKS) is a certified conformant, managed Kubernetes service. Amazon EKS helps you provide highly available and secure clusters and automates key tasks such as patching, node provisioning, and updates.

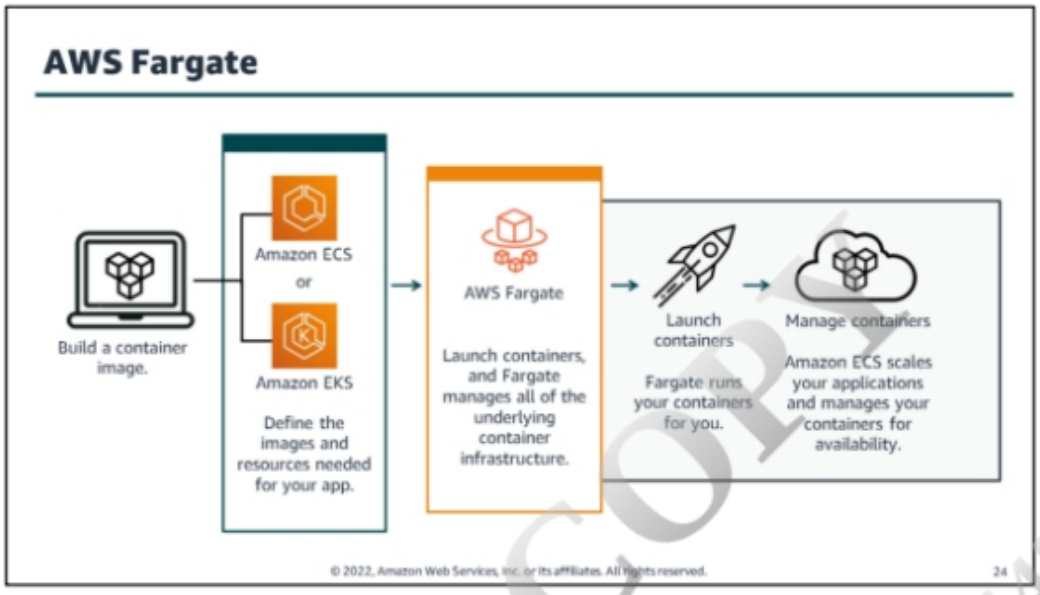
- **Run applications at scale** – Define complex containerized applications and run them at scale across a cluster of servers.
- **Seamlessly move applications** – Move containerized applications from local development to production deployments on the cloud.
- **Run anywhere** – Run highly available and scalable Kubernetes clusters.

For more information, see "Kubernetes on AWS" (https://aws.amazon.com/kubernetes/).



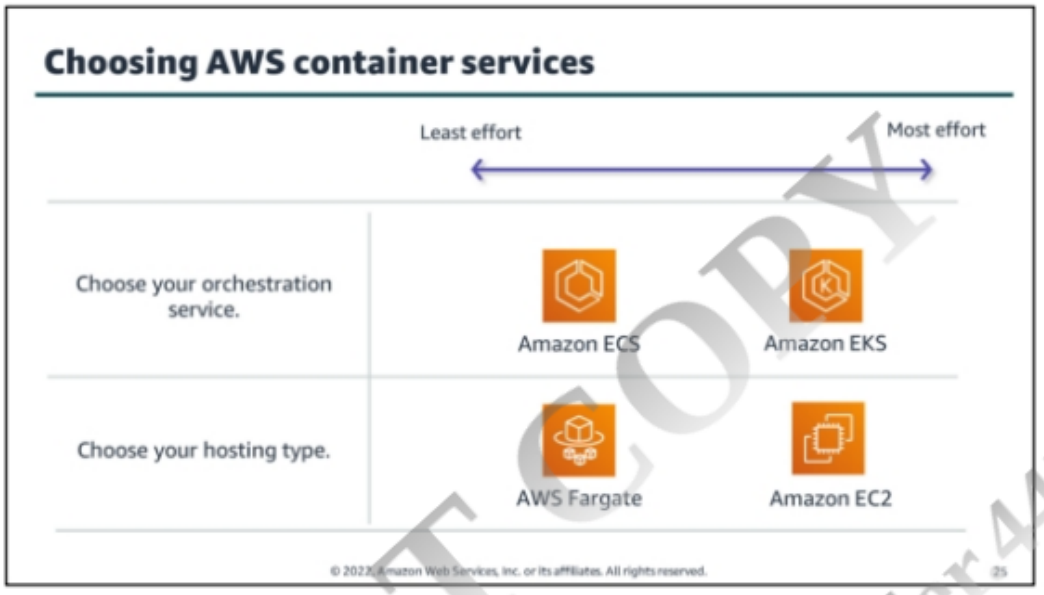
Amazon EKS is a managed service that you can use to run Kubernetes on AWS without having to install and operate your own Kubernetes clusters. With Amazon EKS, AWS manages highly available services and upgrades for you. Amazon EKS runs three Kubernetes managers across three Availability Zones. It detects and replaces unhealthy managers and provides automated version upgrades and patching for the managers. Amazon EKS is also integrated with many AWS services to provide scalability and security for your applications.

Amazon EKS runs the latest version of the open-source Kubernetes software, so you can use all of the existing plugins and tooling from the Kubernetes community. Applications running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, whether running in on-premises data centers or on public clouds.



AWS Fargate is a technology for Amazon ECS and Amazon EKS that you can use to run containers without having to manage servers or clusters. With Fargate, you no longer have to provision, configure, and scale clusters of VMs to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

Fargate eliminates the need for you to interact with or think about servers or clusters. With Fargate, you can focus on designing and building your applications instead of managing the infrastructure that runs them.



Deploying your managed container solutions on AWS involves selecting an orchestration tool and a launch type.

Managing your containerized applications

- Amazon ECS provides a more managed solution with less manual configuration and easier integration with other AWS services.
- Amazon EKS gives you the flexibility to start, run, and scale Kubernetes applications in the AWS cloud or on premises without having to install and operate your own Kubernetes clusters. This is a good option for organizations that work with open source tools. Amazon EKS requires more configuration, but offers more control over your environment.

Container hosting

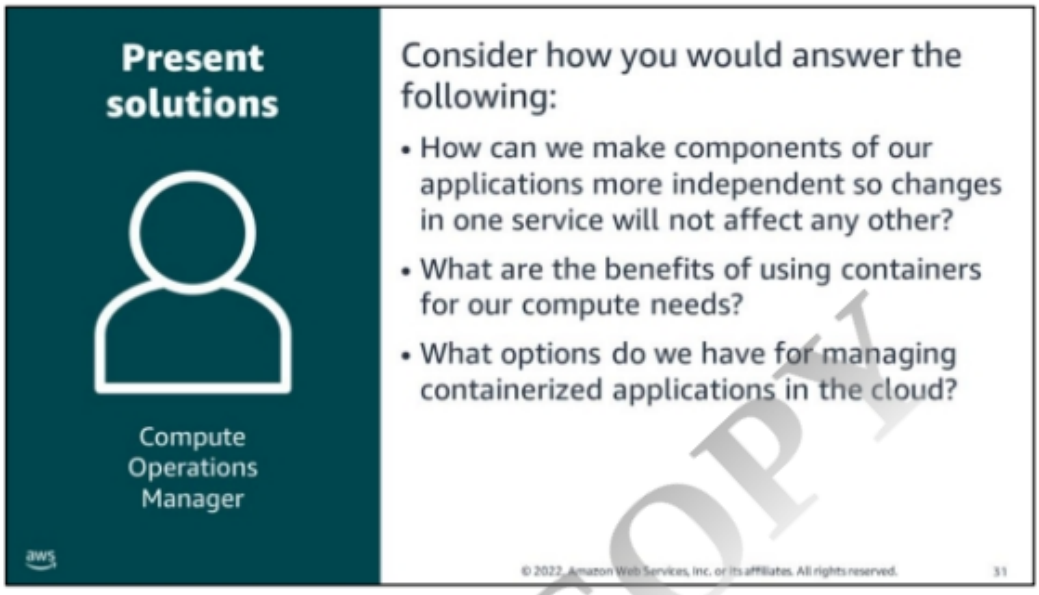
- Hosting on Amazon EC2 requires more configuration, but also provides more control over the resources you use to host your containers. Hosting on Amazon EC2 is also more cost effective.
- Hosting on AWS Fargate uses serverless technology to deliver autonomous container operations. This reduces the time spent on configuration, patching, and security.

For more information about Amazon EKS, see "Amazon EKS features" (https://aws.amazon.com/eks/features/).

For more information about Amazon ECS, see "Amazon Elastic Container Service features" (https://aws.amazon.com/ecs/features/).

For more information about Amazon ECS cluster auto scaling, see "Amazon ECS cluster Auto Scaling" in the *Amazon Elastic Container Service Developer Guide* (https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cluster-auto-scaling.html).

For more information about AWS Fargate, see "AWS Fargate" (https://aws.amazon.com/fargate/).

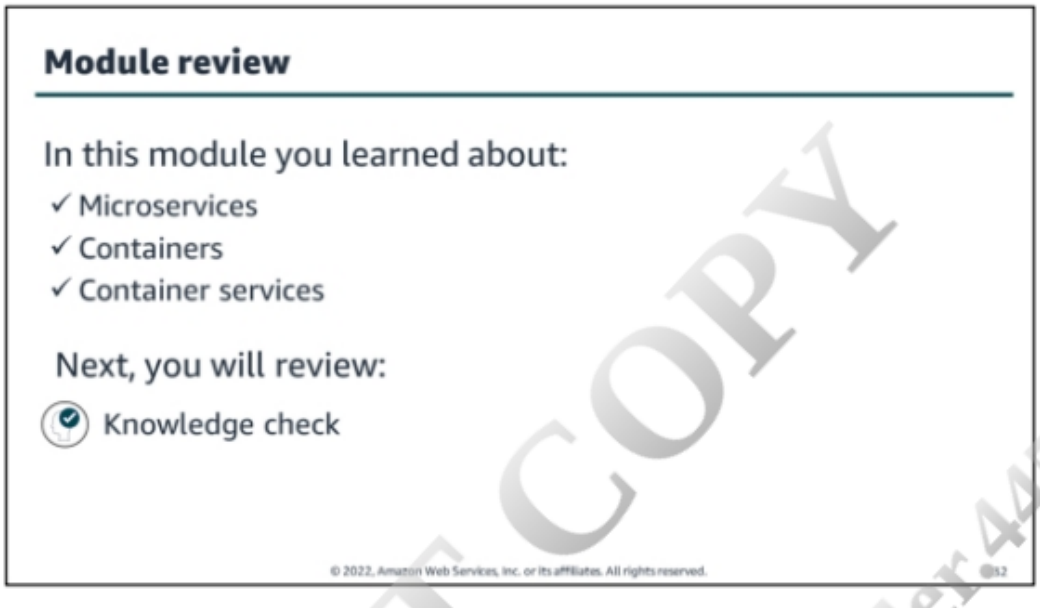


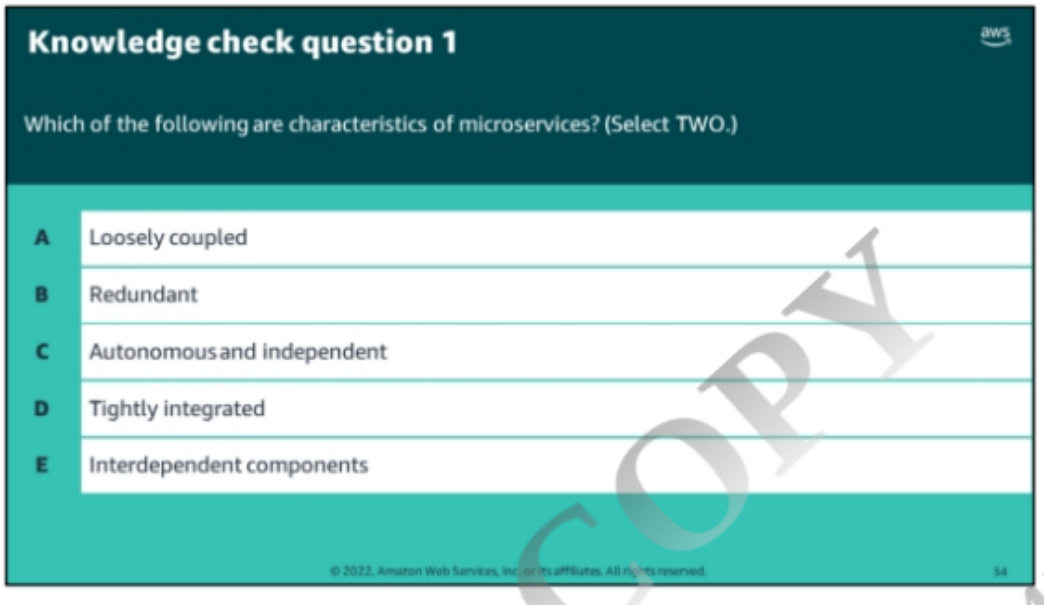
Imagine you are now ready to talk to the compute operations manager and present solutions that meet their architectural needs.

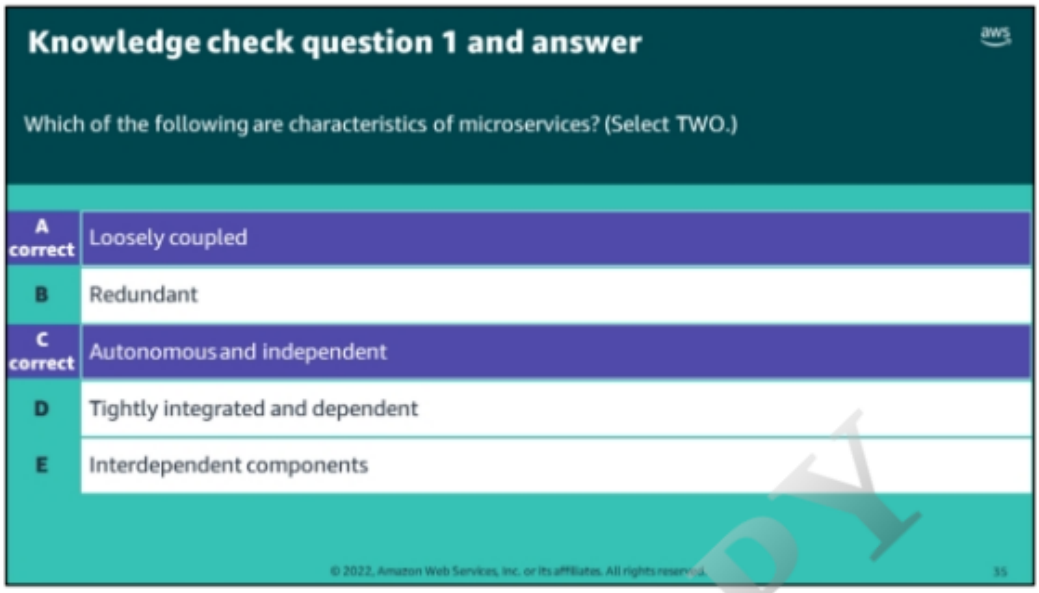
Think about how you would answer the questions from the beginning of the lesson.

Your answers should include the following solutions:

- Using microservice architectures, you can separate your application into component services that can be developed, deployed, operated, and scaled without affecting the other services. A failure in one service is isolated to that service.
- Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. This creates a consistent environment for your applications regardless of the underlying hardware. Containers share an operating system installed on the server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of the environment.
- You can use Amazon ECR as a repository for your container images. You can use Amazon EKS for container orchestration if you use Kubernetes. You can use Amazon ECS for container orchestration for easy integrations with other AWS services. You can host your containers on Amazon EC2 or you can choose AWS Fargate to use serverless container hosting.



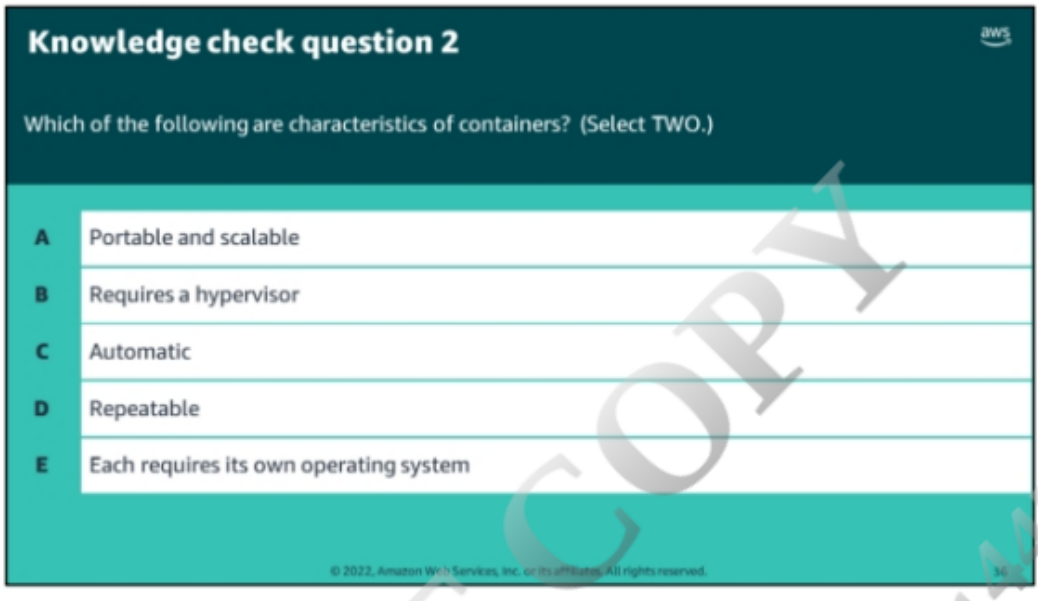


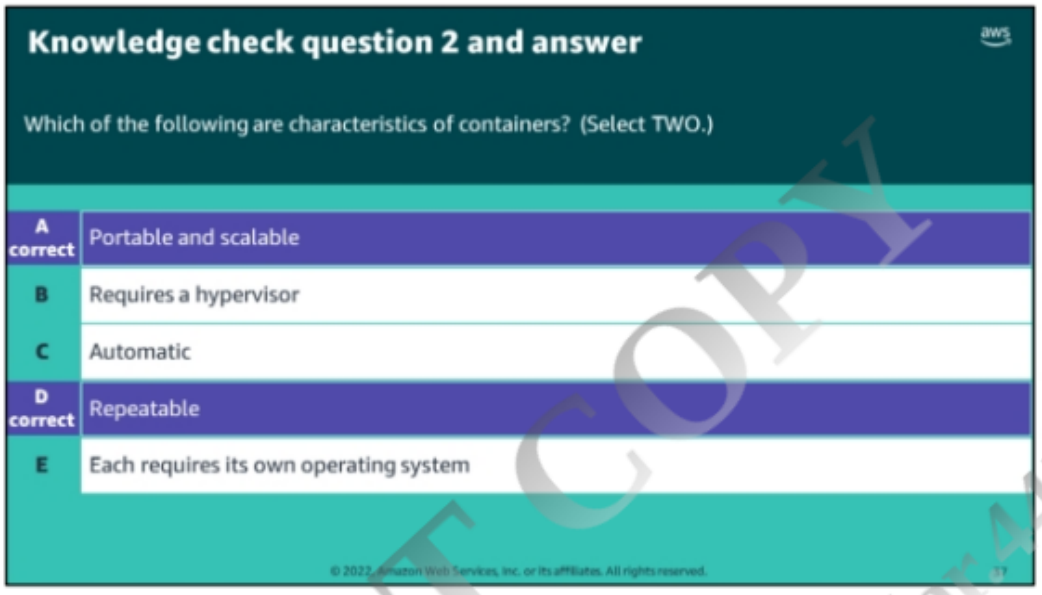


The correct answers are A, loosely coupled, and C, autonomous and independent.

Microservices are loosely coupled. Failures and scaling are automatically handled by the intermediary, such as a load balancer or message queues.

Microservices are autonomous and independent. Each component service in a microservices architecture can be developed, deployed, operated, and scaled without affecting the other services. Services do not need to share any of their code or implementation with other services. Any communication between individual components happens through well-defined APIs.

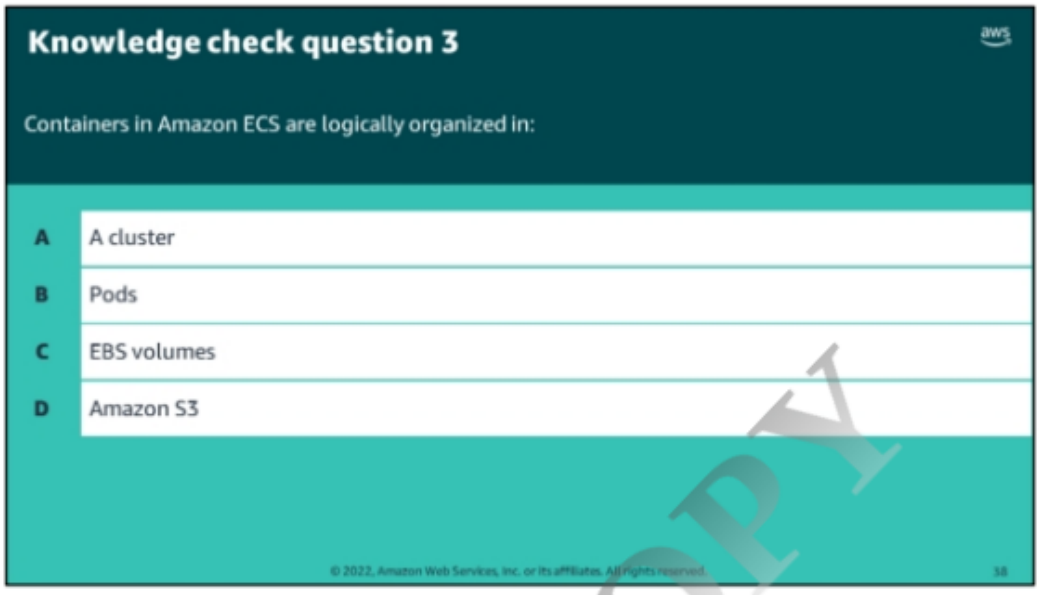


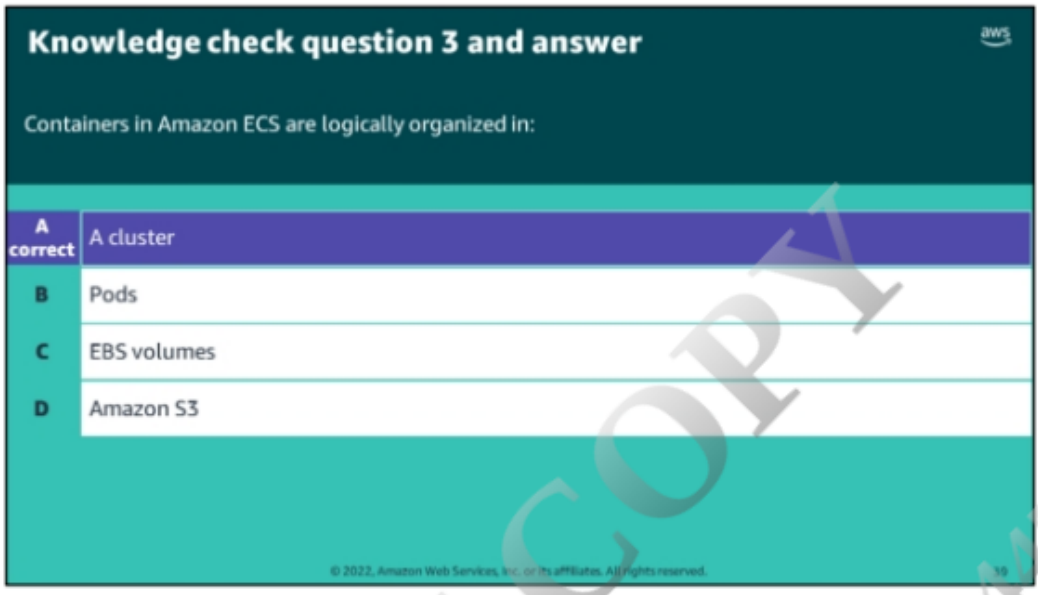


The correct answers are A, portable and scalable, and D, repeatable.

Containers can run on any Linux or Windows system with appropriate kernel-feature support and the container runtime daemon present. This makes them portable. Your laptop, your VM, your Amazon EC2 instance, and your bare metal server are all potential hosts.

Containers are also self-contained environments. Regardless of where you deploy them, the underlying requirements are present and provide uniform behavior.





The correct answer is A, a cluster.

An Amazon ECS cluster is a logical grouping of tasks or services. Your tasks and services are run on infrastructure that is registered to a cluster.

For more information about ECS clusters, see "Amazon ECS Clusters" in the *Amazon Elastic Container Service Developer Guide* (https://docs.aws.amazon.com/AmazonECS/latest/developerguide/clusters.html).

The correct answer is B, to avoid manual infrastructure updates.

With Fargate, you avoid manual infrastructure updates. You don't have to manage servers or clusters of Amazon EC2 instances. With Fargate, you no longer provision, configure, or scale clusters of virtual machines to run containers.