AWS Academy Cloud Developing

Module 8: Introducing Containers and Container Services

Module 8: Introducing Containers and Container Services

Section 1: Introduction

Module objectives

At the end of this module, you should be able to do the following:

- Describe the history, technology, and terminology behind containers
- Differentiate containers from bare-metal servers and virtual machines (VMs)
- Illustrate the components of Docker and how they interact
- Identify the characteristics of a microservices architecture
- Recognize the drivers for using container orchestration services and the AWS services that you can use for container management
- Host a dynamic website by using Docker containers
- Describe how AWS Elastic Beanstalk is used to deploy containers

Module overview

Sections

1. Introduction
2. Introducing containers
3. Introducing Docker containers
4. Using containers for microservices
5. Introducing AWS container services
6. Deploying applications with Elastic Beanstalk

Lab

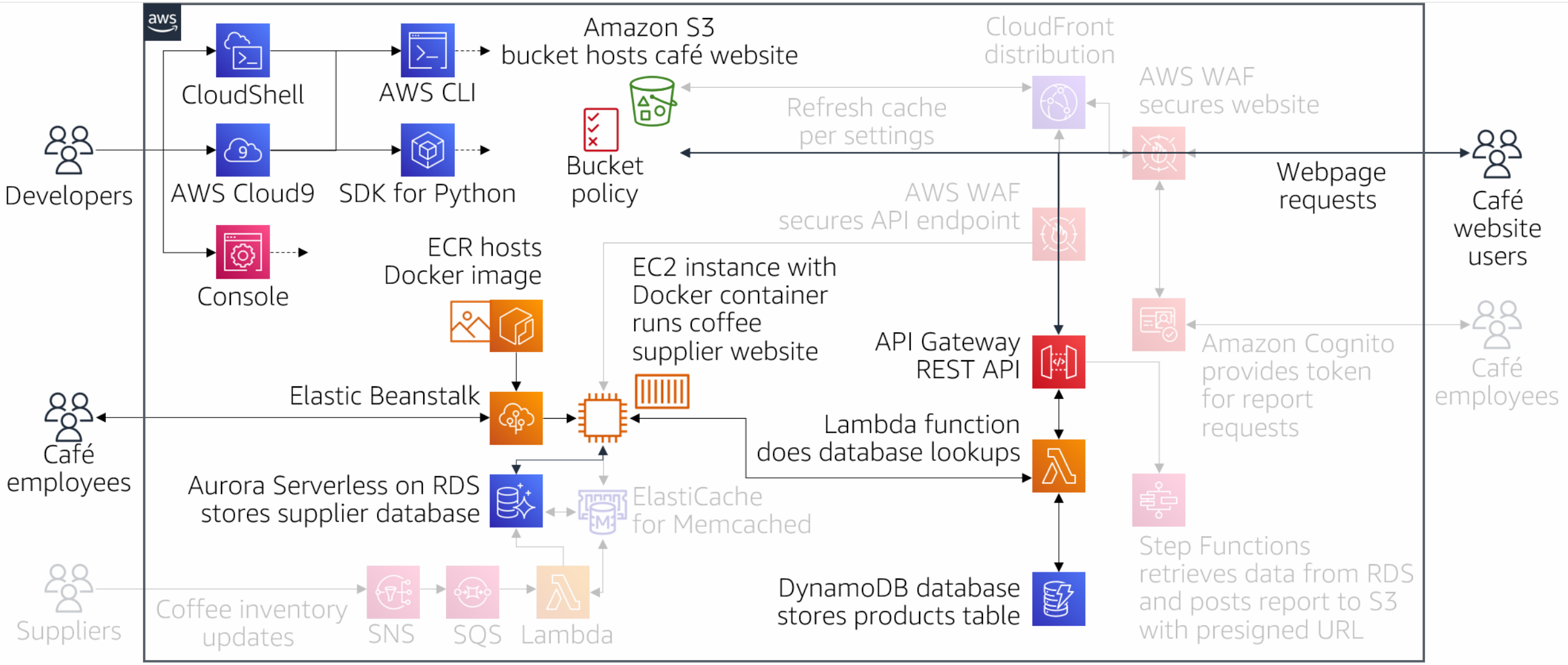
- Lab 1: Migrating a Web Application to Docker Containers
- Lab 2: Running Containers on a Managed Service

Knowledge check

Café business requirement

Frank and Martha recently acquired a coffee bean supplier, and they would like to include the supplier's inventory tracking system into the café's application infrastructure. Sofía is thinking about migrating the application database to containers to complete the integration.

Containers as part of developing a cloud application



Module 8: Introducing Containers and Container Services

Section 2: Introducing containers

Shipping containers

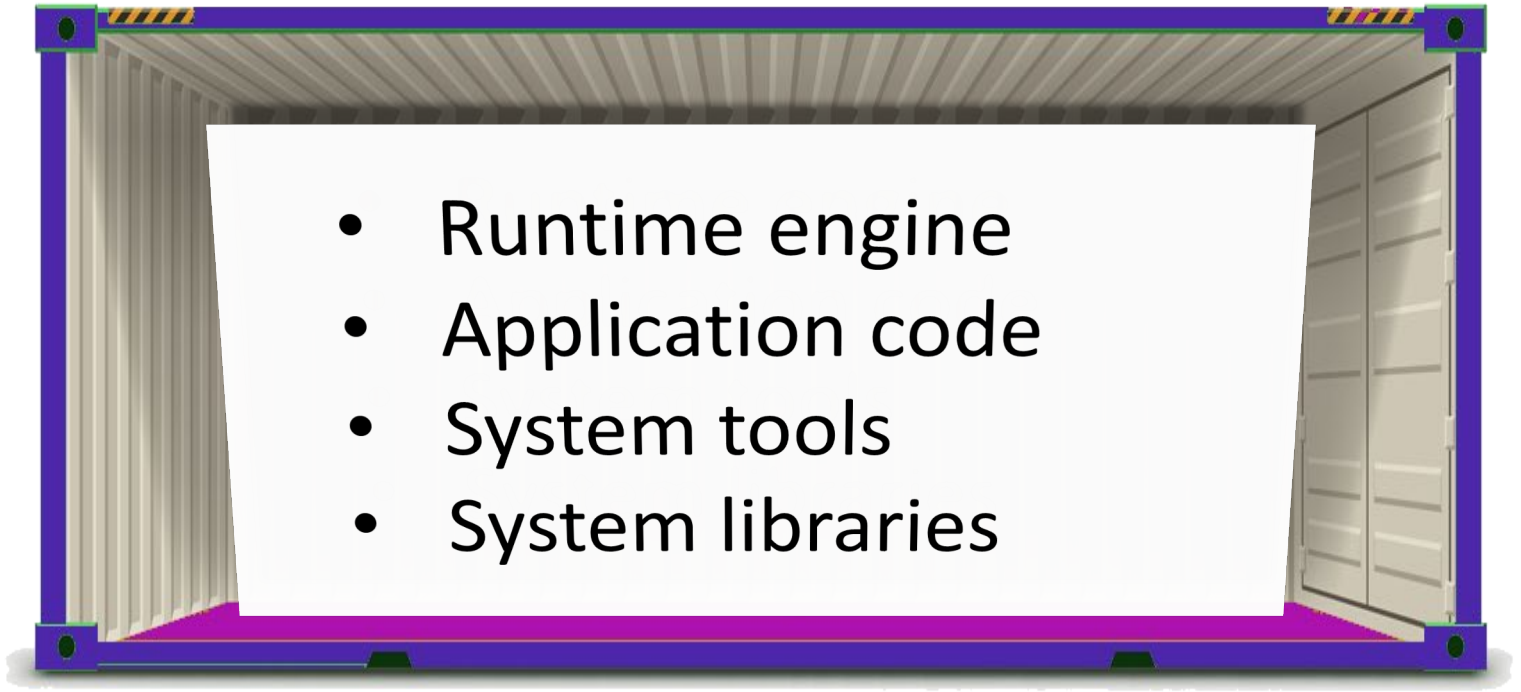
Before shipping containers

- Goods were shipped in a variety of vessels with no standardized weight, shape, or size.
- Transporting goods was slow, inefficient, and costly.

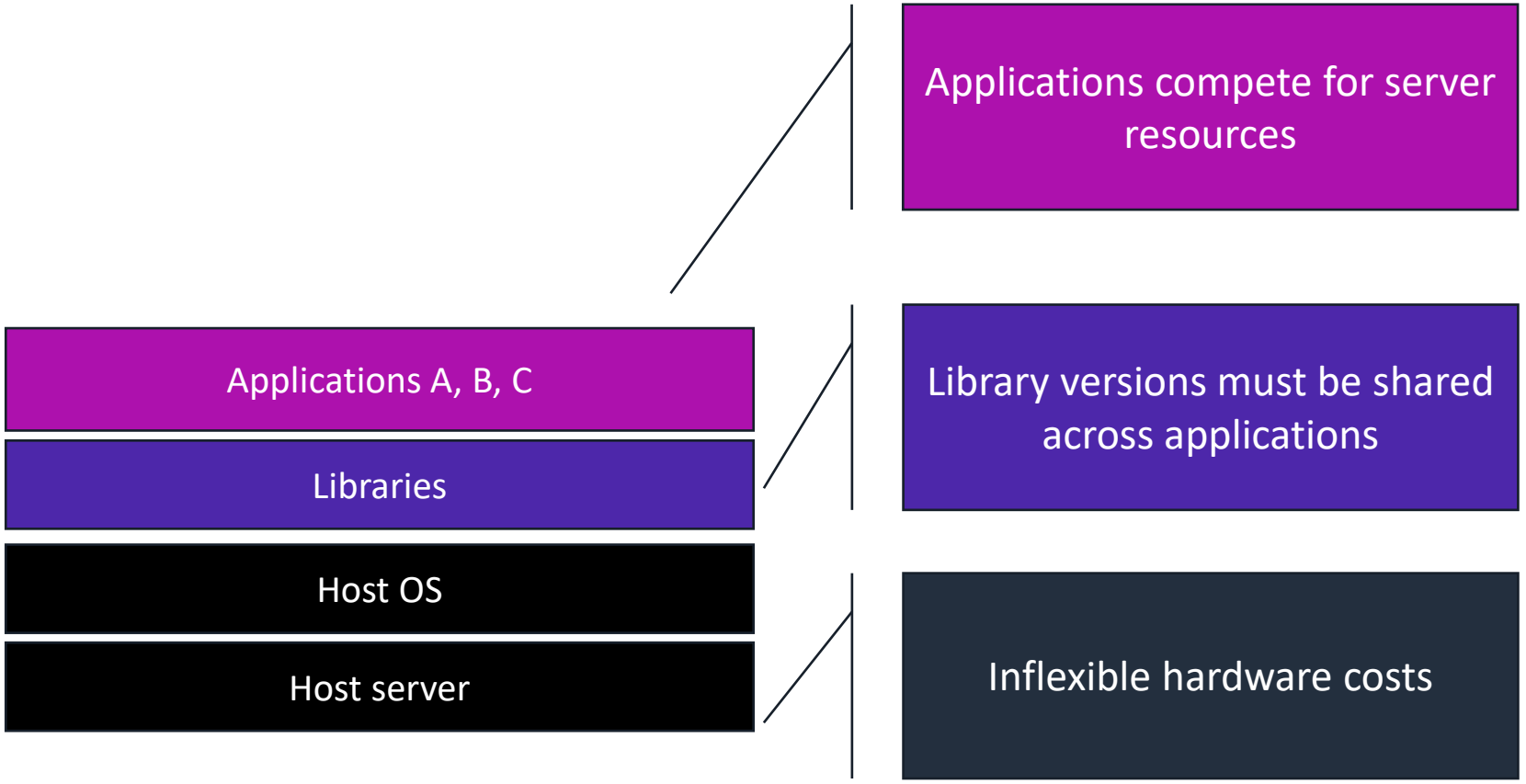
After shipping containers

- Uniformly sized shipping containers simplified loading, unloading, storing, and transferring between transport types.
- Abstraction of shipment details improved efficiency, increased productivity, and reduced costs.

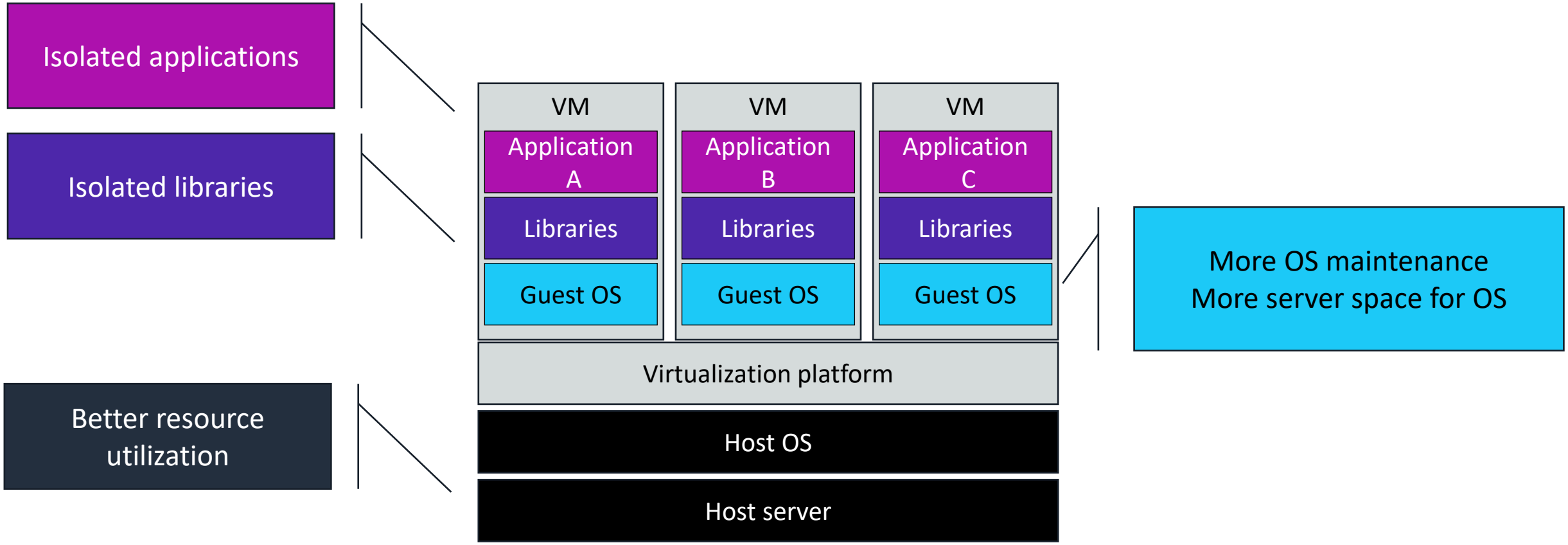
A container is a standardized unit of software



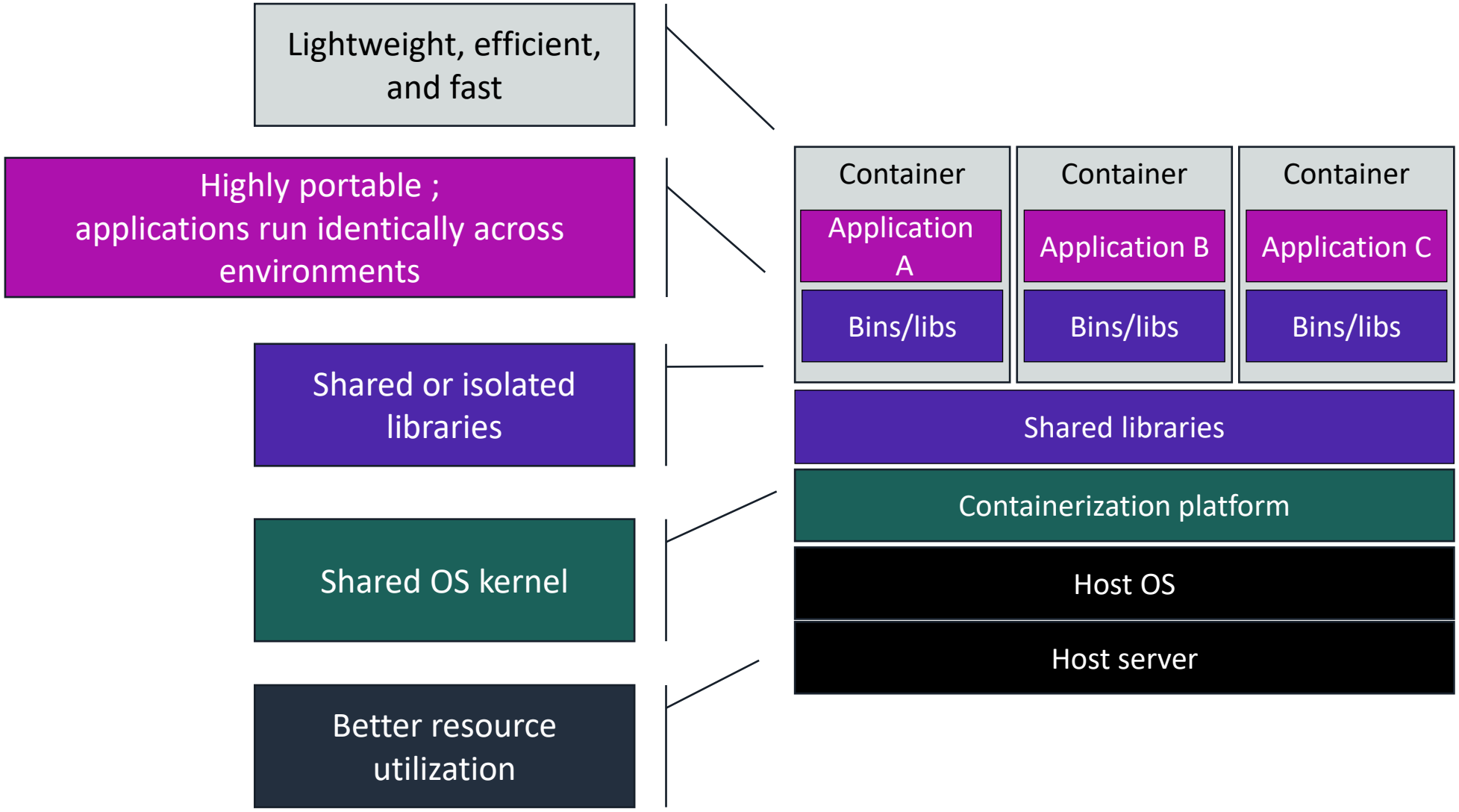
Evolution of deployment models: Bare-metal servers



Evolution of deployment models: VMs



Evolution of deployment models: Containers



Section 2 key takeaways

- A container is a standardized unit of software that contains everything that an application needs to run.
- Containers help to ensure that applications deploy quickly, reliably, and consistently regardless of the deployment environment.

Module 8: Introducing Containers and Container Services

Section 3: Introducing Docker containers

Docker container virtualization platform

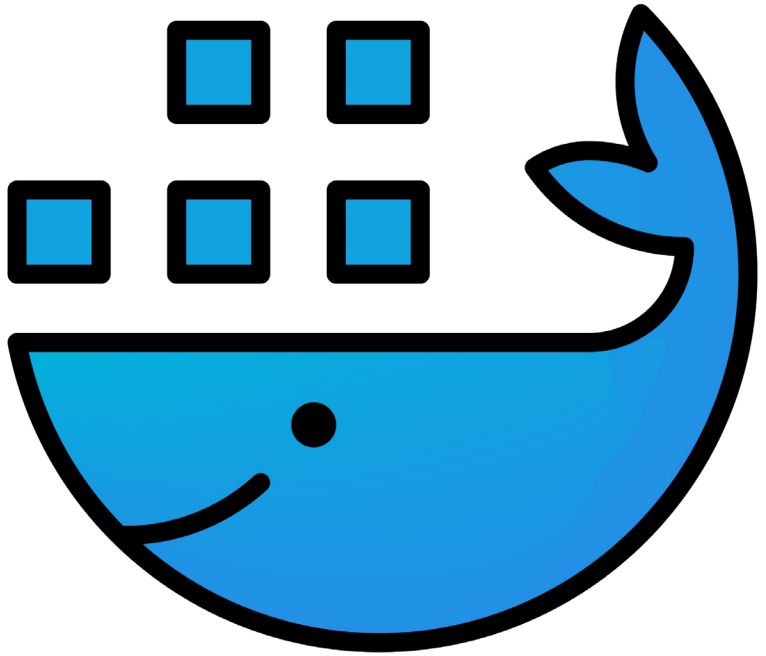
Lightweight container virtualization platform

Tools to create, store, manage, and run containers

Integration with automated build, test, and deployment pipelines

Docker container benefits

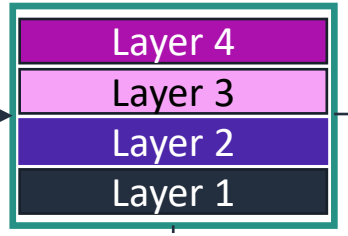
- Portable runtime application environment
- Application and dependencies can be packaged in a single, immutable artifact
- Ability to run different application versions with different dependencies simultaneously
- Faster development and deployment cycles
- Better resource utilization and efficiency



Docker container components

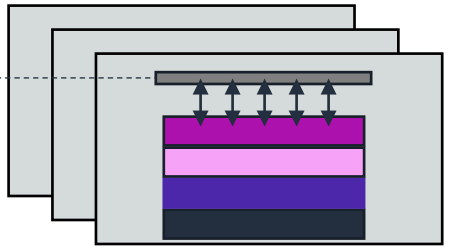
Dockerfile

Plain text file that provides instructions to create a container image



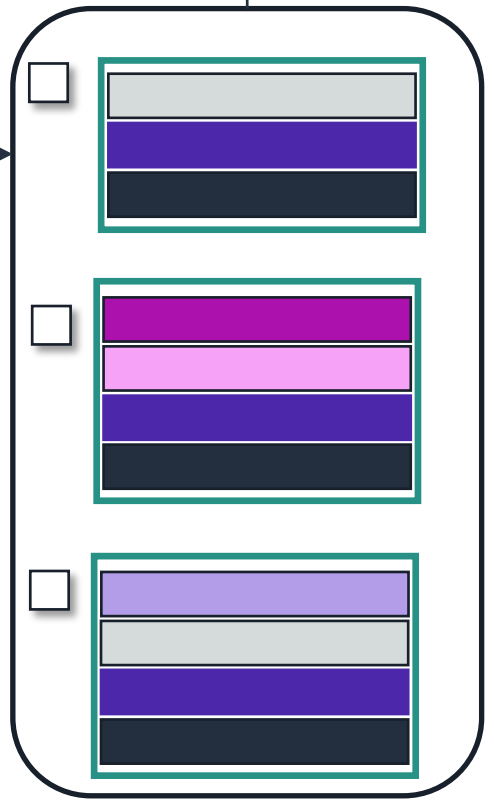
Container image

Read-only template that is used to create writable containers



Container

Runnable instance of an image



Container registry

Private or public images that you can base other images on

Container layer

Thin read/write layer that is used to make changes to the running container

Dockerfile simple example

```
# Start with the Ubuntu latest image
FROM ubuntu:latest

# Output hello world message
CMD echo "Hello World!"
```

Dockerfile example: Start a Java application

```
# Start with open JDK version 8 image
FROM openjdk:8

# Copy the .jar file that contains your
code from your system to the container
COPY /hello.jar /usr/src/hello.jar

# Call Java to run your code
CMD java -cp /usr/src/hello.jar
Org.example.App
```

Dockerfile example: Common tasks

```
# Start with CentOS 7 image
FROM centos:7

# Update the OS and install Apache
RUN yum -y update && yum -y install httpd

# Expose port 80—the port that the web server
“listens to”
EXPOSE Port 80

# Copy shell script and give it run permissions
ADD run-httpd.sh /run-httpd.sh
RUN chmod -v +x /run-httpd.sh

# Run shell script
CMD ["/run-httpd.sh"]
```

Each line of the Dockerfile adds a layer

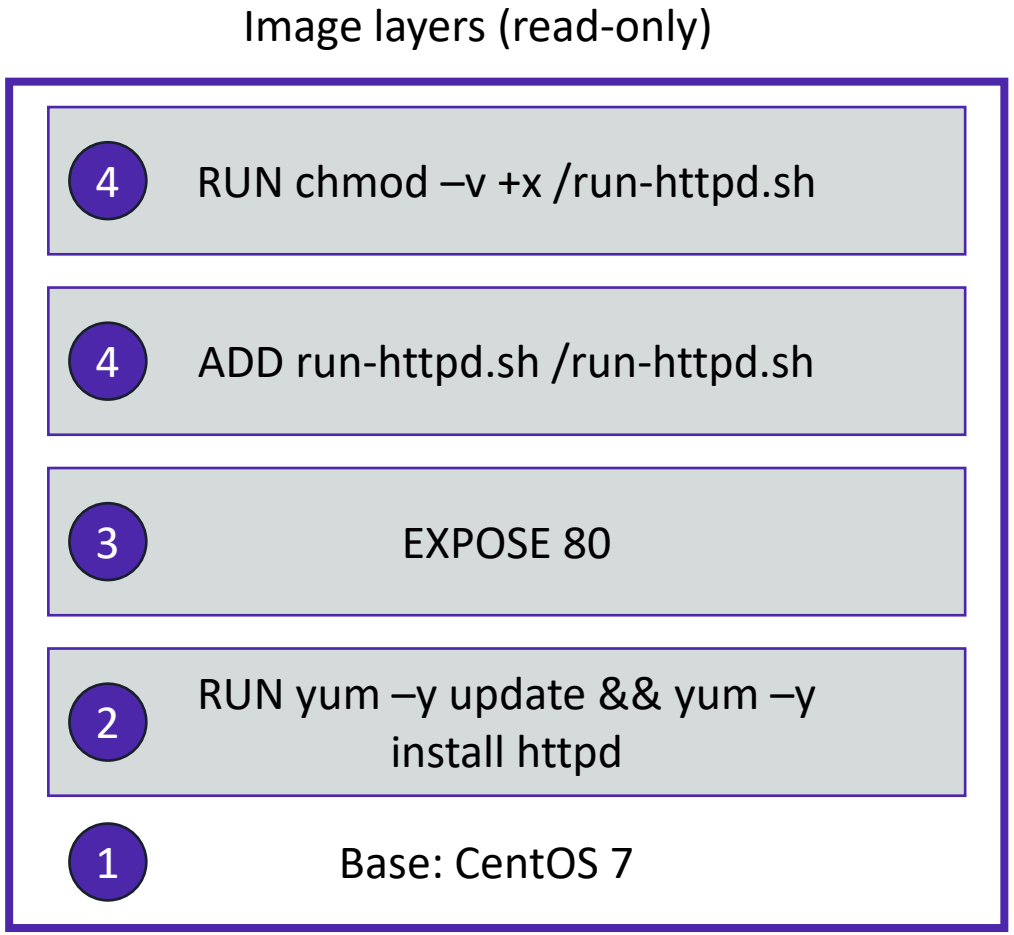
```
# 1 Start with CentOS 7 image
FROM centos:7

# 2 Update the OS and install Apache
RUN yum –y update && yum –y install httpd

# 3 Expose port 80
EXPOSE Port 80

# 4 Copy shell script and give it run
permissions
ADD run-httpd.sh /run-httpd.sh
RUN chmod –v +x /run-httpd.sh

CMD ["/run-httpd.sh"]
```



Docker CLI commands

Command	Description
`docker build`	Build an image from a Dockerfile.
`docker images`	List images on the Docker host.
`docker run`	Launch a container from an image.
`docker ps`	List the running containers.
`docker stop`	Stop a running container.
`docker start`	Start a container.
`docker push`	Push the image to a registry.
`docker tag`	Tag an image.

Command	Description
`docker logs`	View container log output.
`docker port`	List container port mappings.
`docker inspect`	Inspect container information.
`docker exec`	Run a command in a container.
`docker rm`	Remove one or more containers.
`docker rmi`	Remove one or more images from the host.
`docker update`	Dynamically update the container configuration.
`docker commit`	Create a new image from a container's changes.

Example of docker build command

Task	Docker command
Build an image from a Dockerfile in the current directory, and name the image node_app	`docker build --tag node_app .`

Example output

```
Sending build context to Docker daemon  9.007MB

Step 1/7 : FROM node:11-alpine

11-alpine: Pulling from library/node
…

Successfully built a5886f101e12

Successfully tagged node_app:latest
```

Example of docker images command

Task	Docker command
List the images that your Docker client is aware of	`docker images`

Example output

```
REPOSITORY  TAG                 IMAGE ID      CREATED             SIZE

<none>      node_app:latest     a5886f101e12  18 seconds ago      82.7MB
```

Example of docker run command

Tasks	Docker command

- Create a container named `node_app_1` from the image named `node_app`
- Run in the background and print the container ID to the terminal
- Publish container `port 8000` to the host `port 80` to make the container available to other services for HTTP

```
docker run -d --name node_app_1 -p 8000:80 node_app
```

Example output

```
5ed1ea04bcb58194100f71b2e7cd0aecab182313692ed833a6a70064994785f
```

```
docker ps
```

```
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS
5ed1ea04bcb5   node_app   "docker-entrypoint.s…"   9 seconds ago    Up 7 seconds    0.0.0.0:8000->80/tcp
```

Example of docker exec command

Tasks	Docker command

- Start an `sh` terminal session on a running container
- List the files in the `user/src/app` directory
- Exit the shell session on the running container

```
docker exec -it node_app_1 sh
```

Example output

```
/usr/src/app #
```

```
/usr/src/app # ls

Dockerfile          README.md           app
index.js            network.template    node_modules
package-lock.json   package.json        public              views

/usr/src/app # exit
```

Example of docker stop and docker rm commands

Tasks	Docker command
• Stop the container • Remove the container	`docker stop node_app_1 && docker rm node_app_1`

Example output

```
node_app_1
node_app_1
```

Section 3 key takeaways

- Docker containers are created from read-only templates, which are called images.
- Images are built from a Dockerfile and often based on other images.
- Containers are runnable instances of an image with a writable layer.
- A container registry is a repository of images.
- To manage your Docker images and containers, you can run Docker command line interface (CLI) commands from a Bash terminal.

Lab 8.1: Migrating a Web Application to Docker Containers

Lab: Scenario

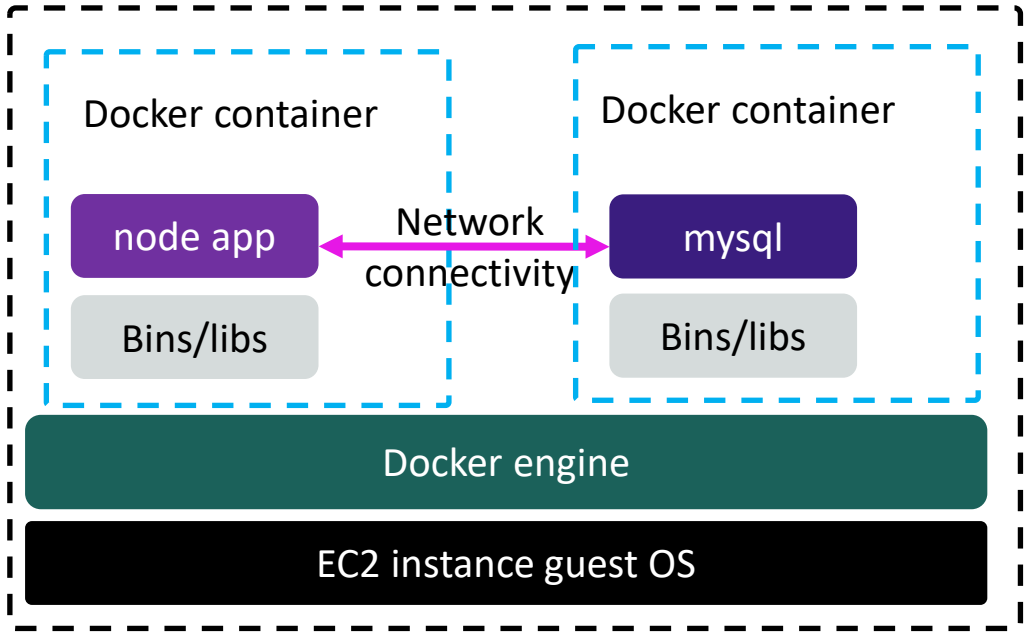
Recently, the café owners acquired one of their favorite coffee suppliers. The acquired coffee supplier runs an inventory tracking application on an AWS account.

In this lab, you again play the role of Sofía, and you will work to migrate the application to run on containers.

Lab: Tasks

1. Preparing the development environment
2. Analyzing the existing application infrastructure
3. Migrating the application to a Docker container
4. Migrating the MySQL database to a Docker container
5. Testing the MySQL container with the node application
6. Adding the Docker images to Amazon ECR

Lab: Final product



AWS Cloud9 EC2 instance (being used as a container host)

~ 90 minutes

Begin Lab 8.1: Migrating a Web Application to Docker Containers

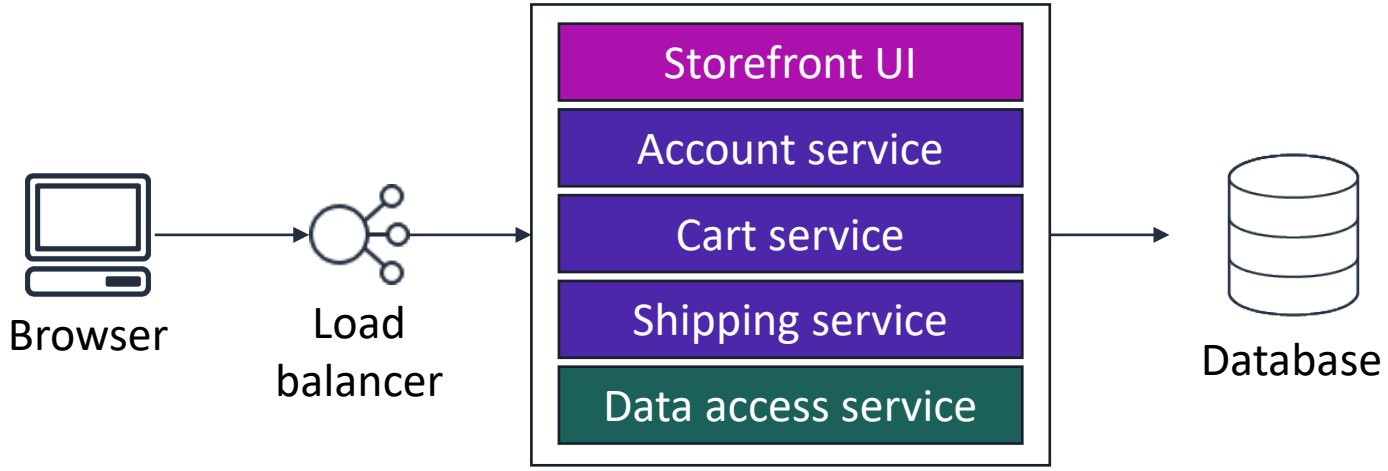
Lab debrief: Key takeaways

Module 8: Introducing Containers and Container Services

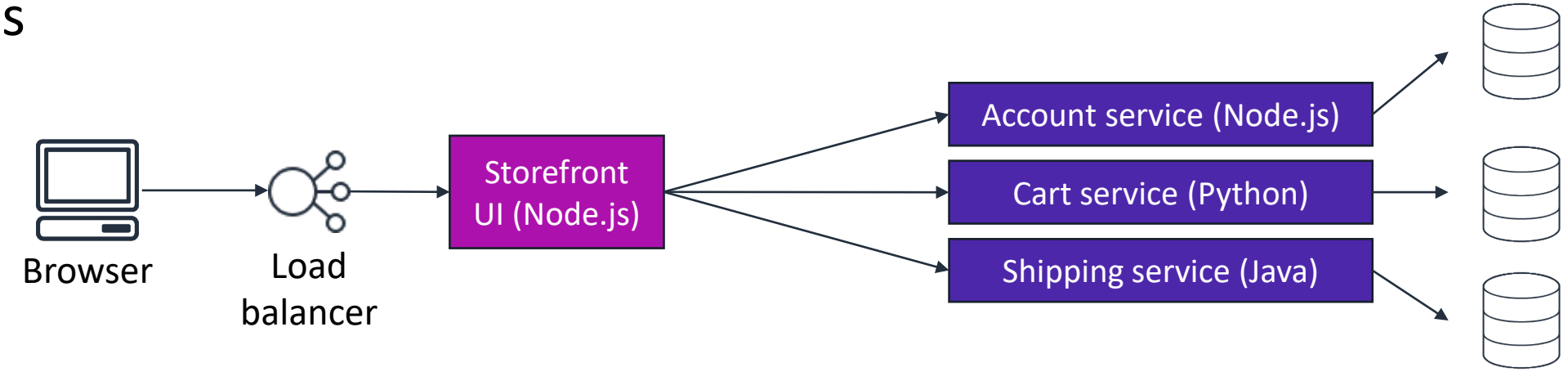
Section 4: Using containers for microservices

Comparing monolithic and microservice architectures

Monolithic



Microservices



Microservices and containers

Microservices design	Container characteristics
• Decentralized, evolutionary design • Smart endpoints, dumb pipes	• Each container uses the language and technology that are best suited for the service. • Each component or system in the architecture can be isolated, and can evolve separately, instead of updating the system in a monolithic style.
• Independent products, not projects	• You can use containers to package all of your dependencies and libraries into a single, immutable object.
• Designed for failure • Disposable	• You can gracefully shut down a container when something goes wrong and create a new instance. You start fast, fail fast, and release any file handlers. • The development pattern is like a circuit breaker. Containers are added and removed, workloads change, and resources are temporary because they constantly change.
• Development and production parity	• Containers can make development, testing, and production environments consistent. • This consistency facilitates DevOps, in which a containerized application that works on a developer's system will work the same way on a production system.

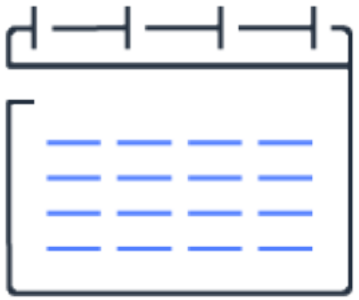
Module 8: Introducing Containers and Container Services

Section 5: Introducing AWS container services

Challenges of managing containers at scale

- State of containers
- Scheduling of starts and stops
- Resources available on each server
- Maximizing availability, resilience, and performance

Container orchestration platforms



Scheduling

Placement

Service integration

Amazon ECS

Amazon Elastic Container Service (Amazon ECS)

Fully managed container orchestration service

- Scales rapidly to thousands of containers with no additional complexity
- Schedules placement across managed clusters
- Integrates with third-party schedulers and other AWS services

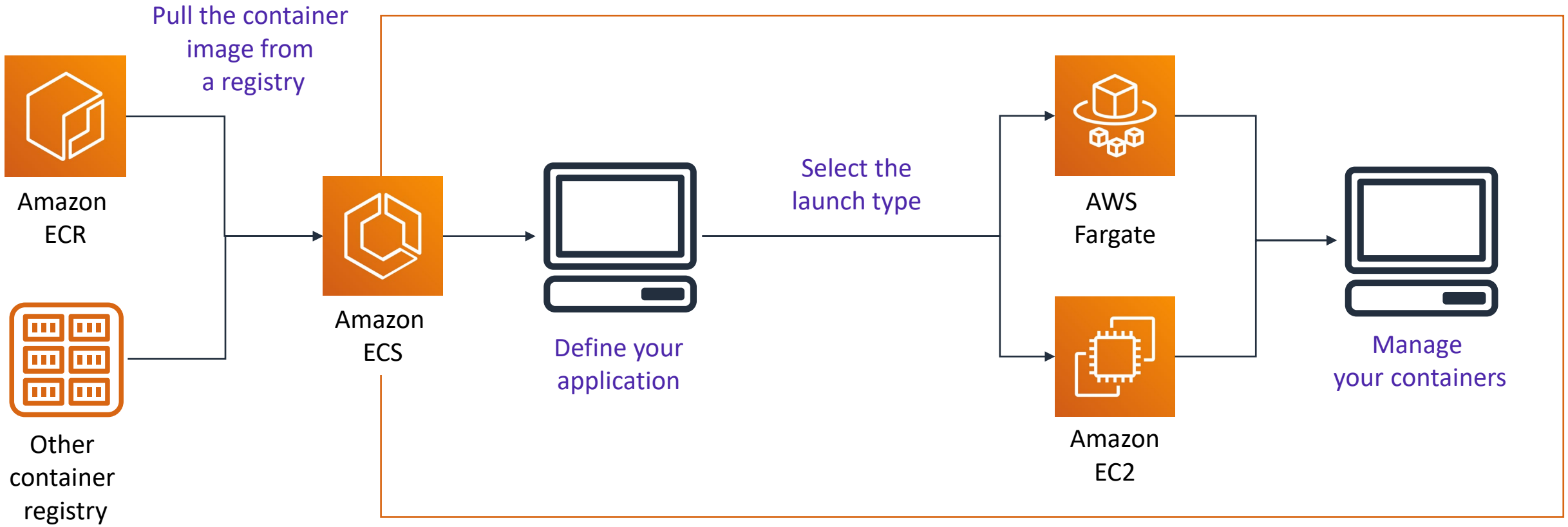
Amazon ECR

Amazon Elastic Container Registry (Amazon ECR)

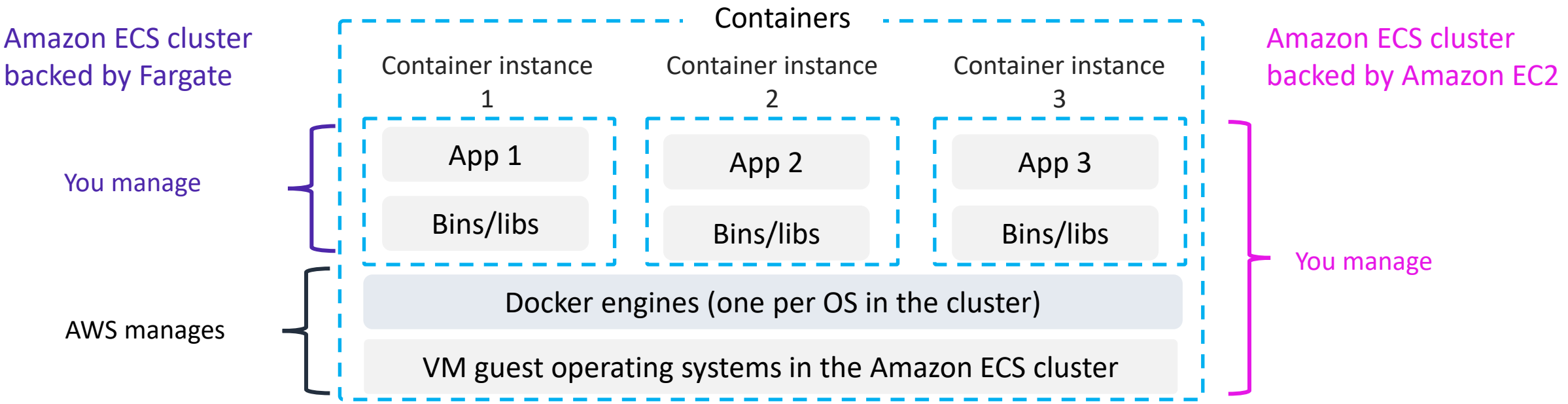
Fully managed container registry that you can use to easily store, run, and manage container images for applications that run on Amazon ECS

- Scalable and highly available
- Integrated with Amazon ECS and Docker CLI
- Secure:
  - Encryption at rest
  - Integration with the AWS Identity and Access Management Service (IAM)

Amazon ECS solution architecture



Amazon ECS with Fargate or Amazon EC2



Choose Fargate:

- Services subject to wide swings in demand
- Large workloads that are optimized for low overhead
- Small test environments
- Batch workloads that run on a schedule

Choose Amazon EC2:

- More predictable resource requirements, or the option of using reserved instances to reduce costs
- Large workloads that are optimized for price
- Compliance with organizational security requirements
- Excess Amazon EC2 capacity

Creating an Amazon ECR repository and pushing an image

```
# Create a repository called hello-world
> aws ecr create-repository \
    --repository-name hello-world \
    --region us-east-1

# Build and tag an image
> docker build -t hello-world .
> docker tag hello-world:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest

# Authenticate Docker to your Amazon ECR registry
# You can skip the `docker login` step if you have amazon-ecr-credential-helper set up
> aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com

# Push an image to your repository
> docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

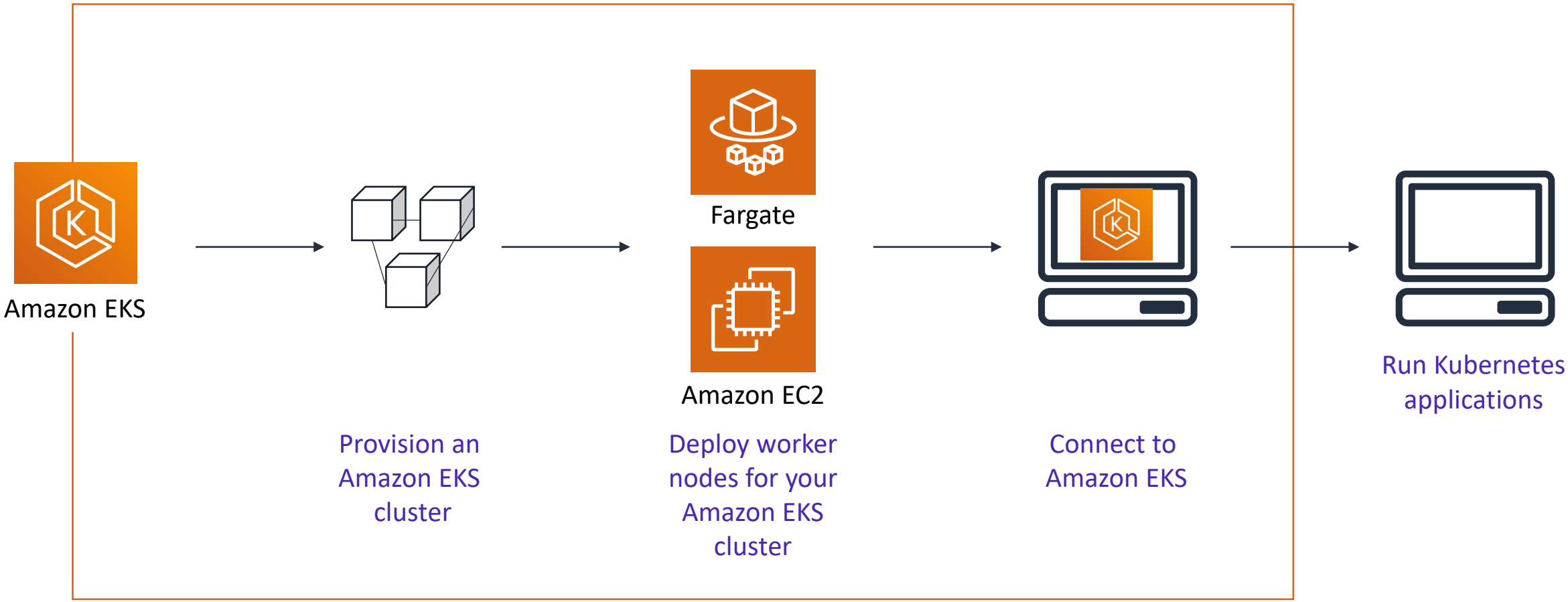
Amazon EKS

Amazon Elastic Kubernetes Service (Amazon EKS)

Managed service that runs Kubernetes on the AWS Cloud

- Built with the Kubernetes community
- Conformant and compatible
- Secure by default

Amazon EKS



Section 5 key takeaways

- Container orchestration services (or systems) simplify managing containers at scale.
- Amazon ECS is a fully managed container orchestration service that you can use to launch containers to either Fargate or EC2 instances.
- Amazon ECR is a fully managed container registry service.
- Amazon EKS is a managed service that you can use to run Kubernetes in the cloud.

Module 8: Introducing Containers and Container Services

Section 6: Deploying applications with Elastic Beanstalk

Elastic Beanstalk

AWS Elastic Beanstalk

Service for deploying and scaling web applications and services

- Automatically handles deployment details like capacity provisioning, load balancing, automatic scaling, and application health monitoring
- Provides a variety of platforms on which to build your applications
- Use to manage all of the resources that run your application as an environment

Elastic Beanstalk components

Component	Description
Application	Logical collection of Elastic Beanstalk components. Conceptually similar to a folder.
Application version	Specific, labeled iteration of deployable code for a web application.
Environment	Collection of AWS resources that run an application version.
Environment tier	Designation of the type of application that the environment runs. Determines what resources Elastic Beanstalk provisions to support it.
Environment configuration	Collection of parameters and settings that define how an environment and its associated resources behave.
Saved configuration	Template that you can use as a starting point for creating unique environment configurations.
Platform	Combination of an OS, programming language runtime, web server, application server, and Elastic Beanstalk components. You design and target your web application to a platform.
Elastic Beanstalk CLI	CLI for Elastic Beanstalk. Provides interactive commands that simplify creating, updating, and monitoring environments from a local repository.

IAM permissions in Elastic Beanstalk environments

IAM roles assigned during environment creation

Service role

- Assigned during creation
- Elastic Beanstalk assumes that it uses other services on your behalf
- Default service role: `aws-elasticbeanstalk-service-role`

Instance profile

- Assigned during creation
- Applied to instances that are launched in your environment
- Default instance profile: `aws-elasticbeanstalk-ec2-role`

User policies

- Optionally assigned
- Can be attached to users or groups who create and manage Elastic Beanstalk applications and environments
- Two managed user policies are available to grant either full administrative access or read-only access

Service role policy example

AWSElasticBeanstalkEnhancedHealth

```
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetHealth",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:GetConsoleOutput",
                "ec2:AssociateAddress",
                "ec2:DescribeAddresses",
                "ec2:DescribeSecurityGroups",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:DescribeNotificationConfigurations",
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
```

Elastic Beanstalk simplifies container deployment

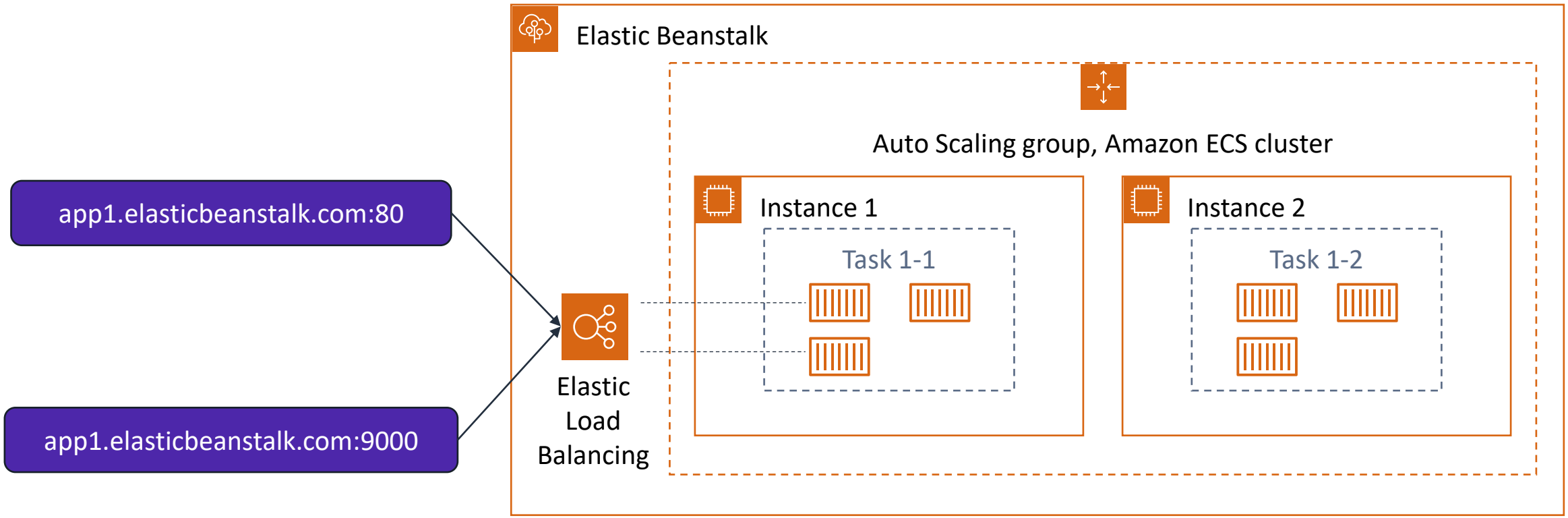
Getting started with Amazon ECS

1. Create a task definition
2. Create and configure a cluster including:
   - EC2 instances
   - VPC settings
   - IAM role definition
3. Create a service to run and maintain a specified number of instances of a task

Getting started with Elastic Beanstalk

1. Write a Dockerrun.aws.json file and provide your zipped code
2. Select the platform for your language
3. Launch your application

Multicontainer Docker platform



Dockerrun.aws.json file

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath":
"/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath":
"/var/app/current/proxy/conf.d"
      }
    }
  ]

 …
```

```
 "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "environment": [
        {
          "name": "Container",
          "value": "PHP"
        }
      ],
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    }
…
```

Elastic Beanstalk deployment policies

Faster → More control

All at once

- Deploys the new version to each instance
- Requires some downtime
- Quickest deployment method

Rolling

- Deploys to a batch of instances at a time
- Avoids downtime
- Minimizes reduced availability
- Longer deployment

Rolling with batch

- Launches an extra batch of instances, then performs a rolling deployment
- Avoids reduced availability
- Longer deployment than rolling

Immutable

- Launches a second Auto Scaling group, and serves traffic to both old and new versions until the new instances pass health checks
- Ensures that the new version always goes on new instances
- Allows for quick and safe rollback
- Longer deployment time

Traffic splitting

- Launches a full set of new instances like an immutable deployment
- Tests the health of the new version by using a portion of traffic while keeping the rest of the traffic served by the old version
- Supports canary testing
- No interruption of service if you must roll back

Deployment option namespaces

`aws:elasticbeanstalk:command`

- Choose the deployment policy
- Set a timeout
- Choose options for size and type of batches to use
- Choose whether to cancel deployment on a failed health check

`aws:elasticbeanstalk:trafficsplitting`

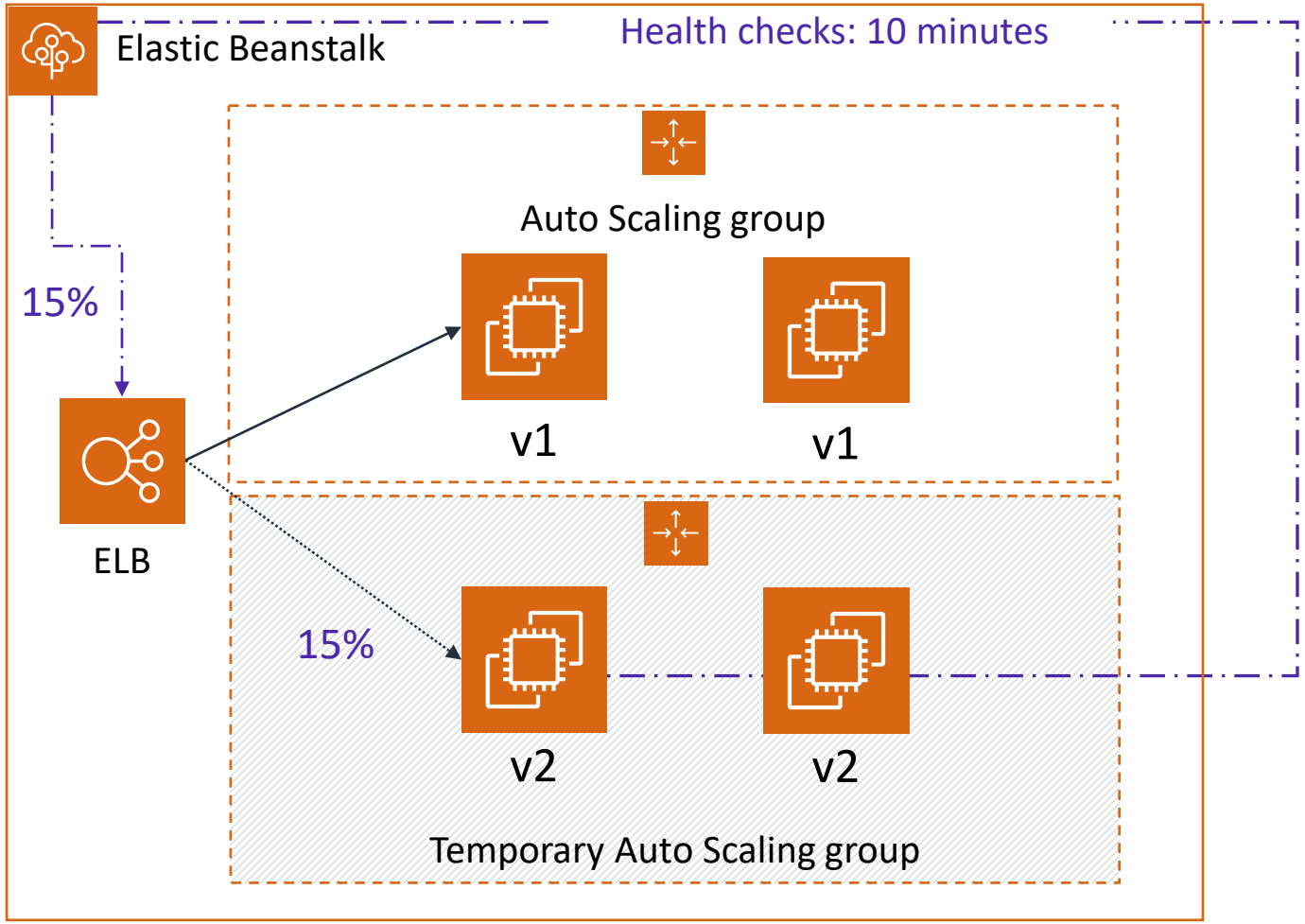
- Choose the percentage of traffic to go to new instances
- Choose how long to wait before continuing to shift more traffic

Example of traffic splitting (canary testing)

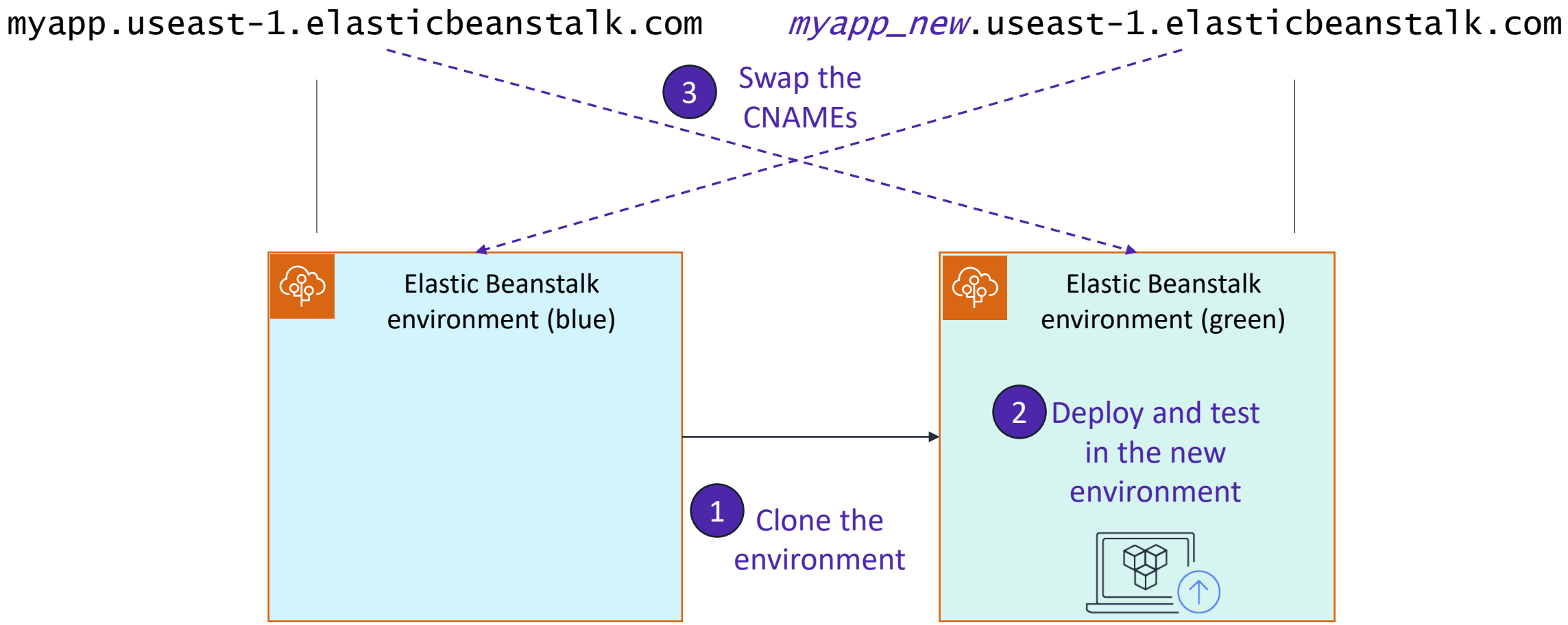
Example deployment configurations

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: TrafficSplitting

aws:elasticbeanstalk:trafficsplitting:
    NewVersionPercent: "15"
    EvaluationTime: "10"
```



Blue/green deployments on Elastic Beanstalk



Section 6 key takeaways

- You can use Elastic Beanstalk to manage all of the resources that run your application as an environment.
- You can quickly launch a Docker multicontainer environment with Elastic Beanstalk without worrying about Amazon ECS configuration details.
- Deployment options include traffic splitting and blue/green to support testing new versions.

Lab 8.2: Running Containers on a Managed Service

Lab: Scenario

Sofía has containerized the coffee suppliers application, but wants to reduce the effort to maintain the application and improve its scalability.

As noted in the previous lab, Sofía wants to move the database to a managed database service rather than running it in a container.

Based on her research, she has made these decisions:

- Use AWS Elastic Beanstalk to deploy the application website.
- Use Amazon Aurora Serverless for the database. Sofía must retire the container-based MySQL database and load the required user, tables, and data into an Aurora Serverless database.

Aurora Serverless

Amazon Aurora

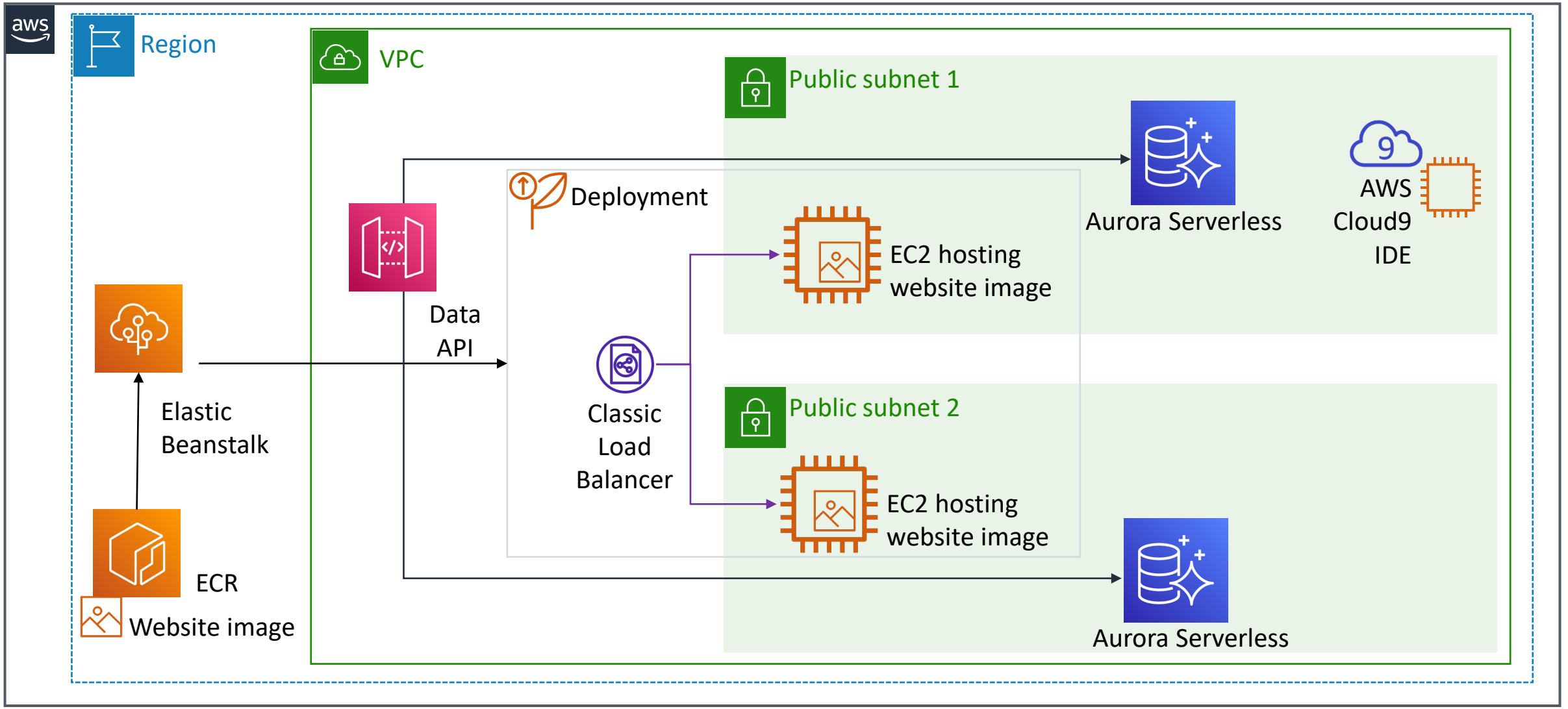
Fully managed relational database engine that is compatible with MySQL and PostgreSQL

- Part of the Amazon Relational Database Service (Amazon RDS), a managed database service
- Combines the performance and availability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases
- Offers Aurora Serverless, an on-demand configuration that automatically scales up or down based on traffic and shuts down when not in use

Lab: Tasks

1. Preparing the development environment
2. Configuring the subnets for Amazon RDS and Elastic Beanstalk to use
3. Setting up an Aurora Serverless database
4. Reviewing the container image
5. Configuring communication between the container and the database
6. Creating the application database objects
7. Seeding the database with supplier data
8. Reviewing the AM policy and role for Elastic Beanstalk
9. Creating an Elastic Beanstalk application
10. Configuring the API Gateway proxy

Lab: Final product



~ 90 minutes

Begin Lab 8.2: Running Containers on a Managed Service

Lab debrief: Key takeaways

Module 8: Introducing Containers and Container Services

Module wrap-up

Module summary

In summary, in this module, you learned how to do the following:

- Describe the history, technology, and terminology behind containers
- Differentiate containers from bare-metal servers and VMs
- Illustrate the components of Docker and how they interact
- Identify the characteristics of a microservices architecture
- Recognize the drivers for using container orchestration services and the AWS services that you can use for container management
- Host a dynamic website by using Docker containers
- Describe how Elastic Beanstalk is used to deploy containers

Complete the knowledge check

Sample exam question

A cloud architect wants to migrate a web application to containers. The team does not have much experience with AWS or containers, but the architect wants to get them started quickly to be able to experiment.

Which solution would be best?

A. Use Elastic Beanstalk to launch a multicontainer Docker environment.

B. Use Amazon ECR to host Docker images that they create from scratch.

C. Configure EC2 instances with automatic scaling, and install Docker images on the instances.

D. Configure Amazon ECS with a cluster of EC2 instances that run Docker containers.

Additional resources

- Blog posts
  - Building Container Images on Amazon ECS on AWS Fargate: https://aws.amazon.com/blogs/containers/building-container-images-on-amazon-ecs-on-aws-fargate/
  - Developing Twelve-Factor Apps Using Amazon ECS and AWS Fargate: https://aws.amazon.com/blogs/containers/developing-twelve-factor-apps-using-amazon-ecs-and-aws-fargate/
- Amazon ECS Workshop: https://ecsworkshop.com/

Thank you

aws academy