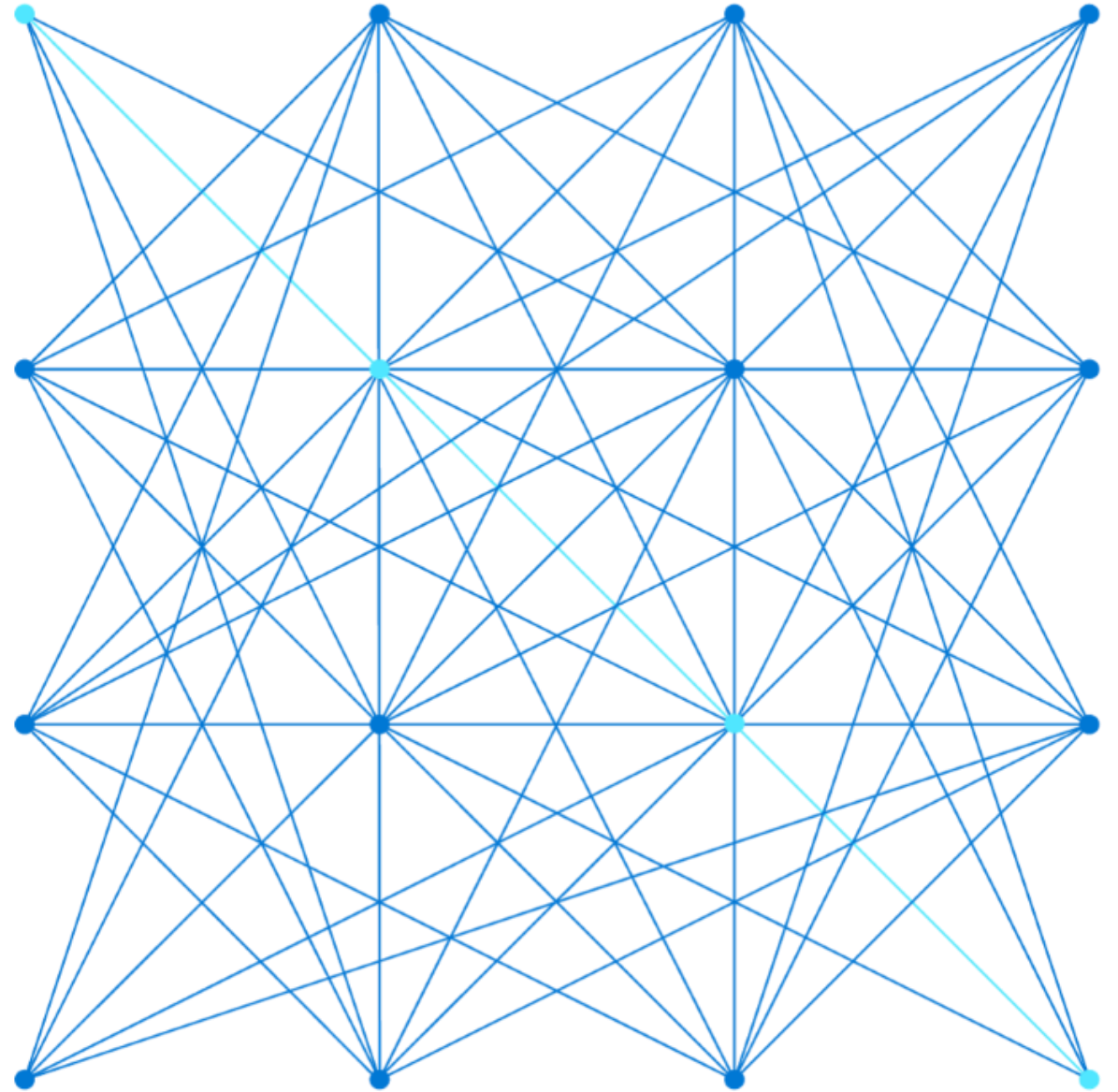


AZ-400.00

Module 15: Managing Containers using Docker



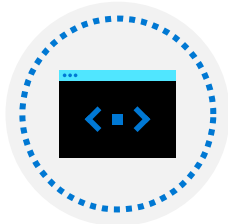
Lesson 01: Module overview



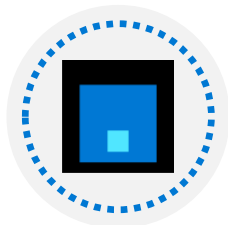
Module overview



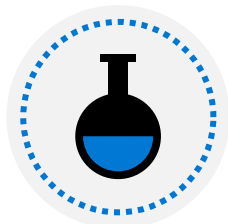
Lesson 1: Module overview



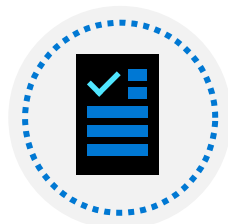
Lesson 2: Implementing a container build strategy



Lesson 3: Implementing Docker multi-stage builds



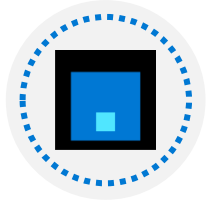
Lesson 4: Lab



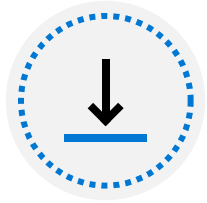
Lesson 5: Module review and takeaways

Learning objectives

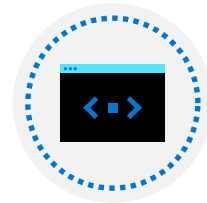
After completing this module, students will be able to:



Implement a container strategy including how containers are different from virtual machines and how microservices use containers

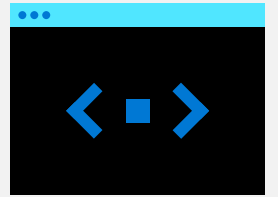


Implement containers using Docker

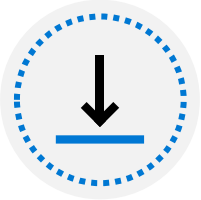


Implement Docker multi-stage builds

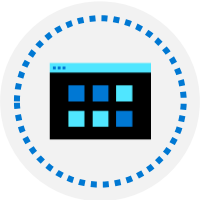
Lesson 02: Implementing a container build strategy



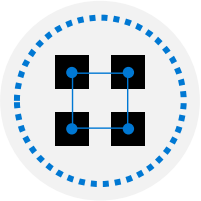
Why containers?



Portable – run containers wherever Docker is supported



Consistent – know that developers are working with identical code

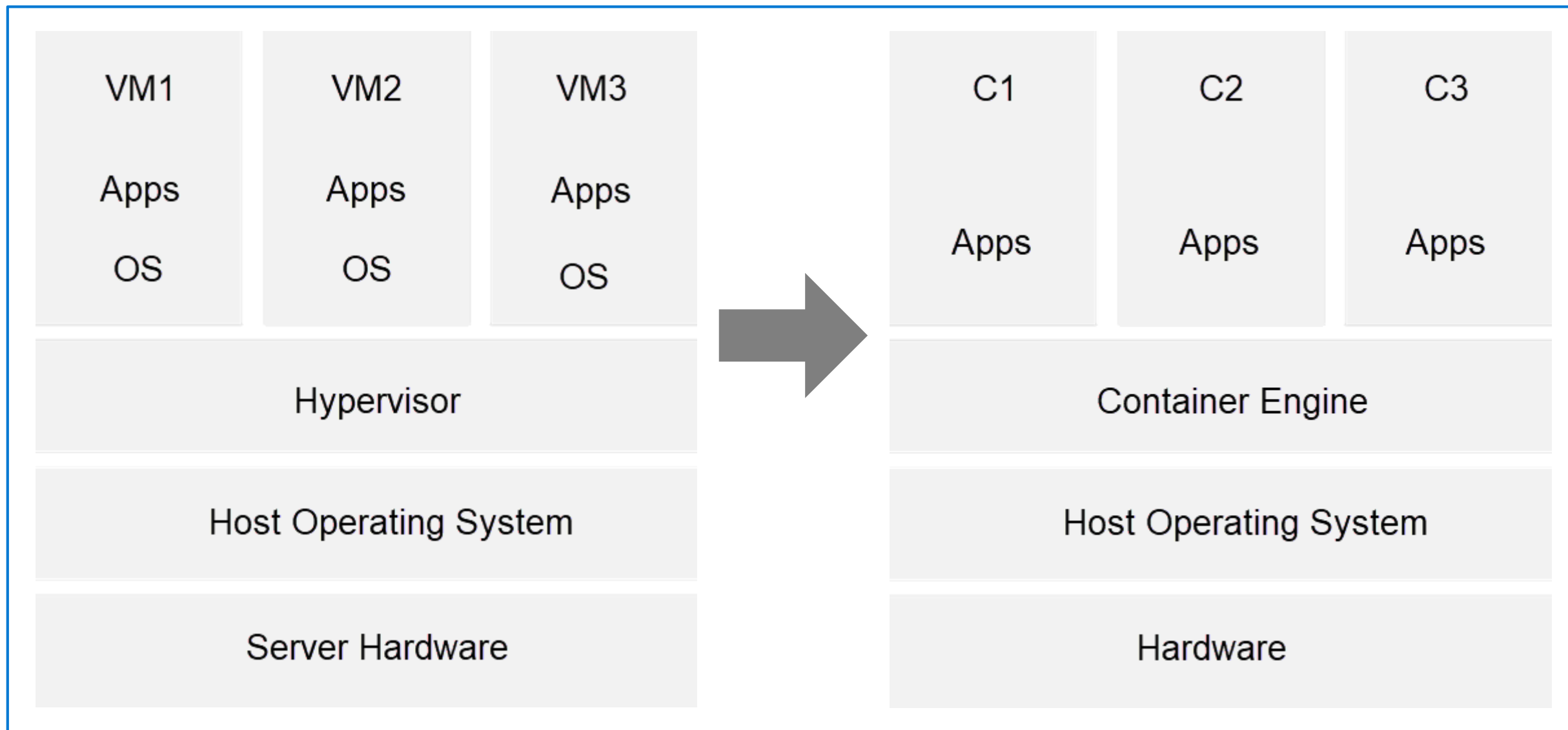


Lightweight – far less disk, CPU, and memory than VMs

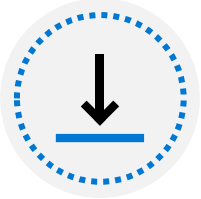


Efficient – fast deployment, booting, patching, updates

Structure of containers



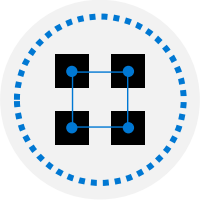
Docker containers and development



Docker is a software containerization platform with a common toolset, packaging model, and deployment mechanism



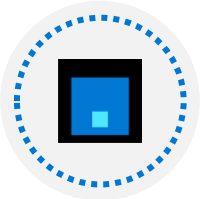
Docker greatly simplifies containerization and distribution of applications that can be run anywhere



A Docker image can be created that will deploy identically across any environment in seconds

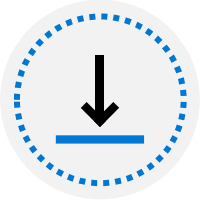


DockerHub has more than 180,000 applications in the public community repository

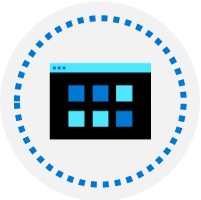


Docker organized the Open Container Initiative (OCI)

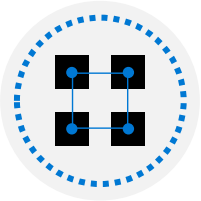
Working with Docker containers



docker build – create the image



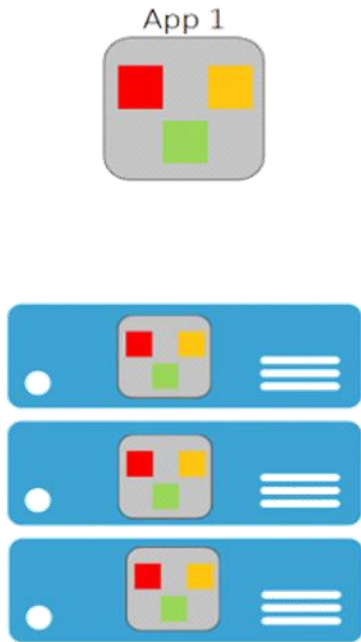
docker pull – retrieve an image from a container registry



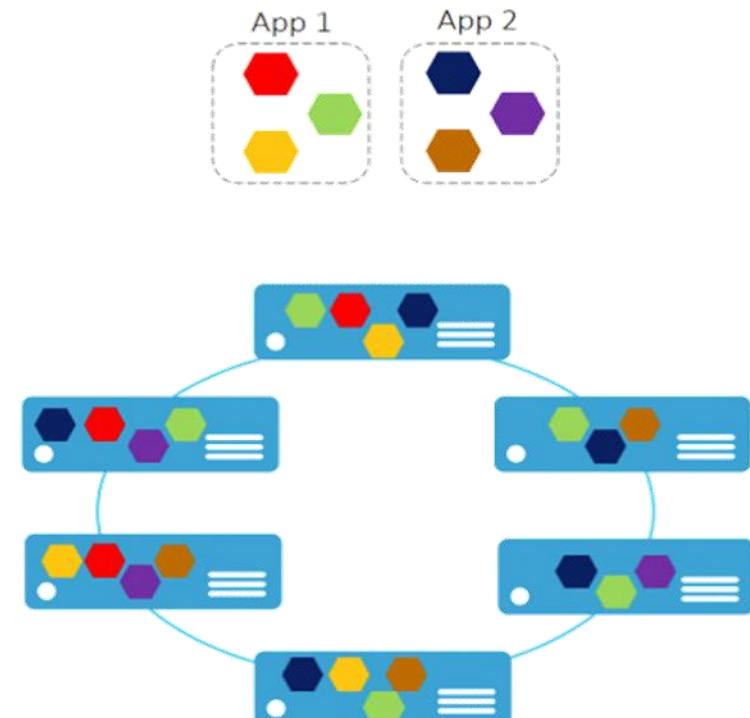
docker run – run the image to create a container instance (will auto pull if not already pulled)

Microservices and containers

Monolithic application approach

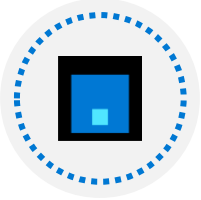


Microservices application approach

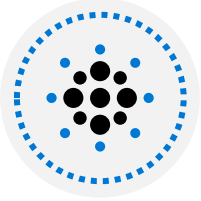


With Microservices every part of the application is deployed as a fully self-contained component

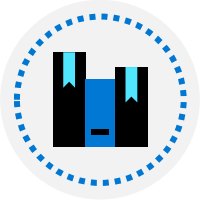
Azure container-related services



Azure Container Instances let you focus on creating your applications rather than provisioning and management of the infrastructure



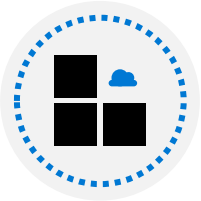
Azure Kubernetes Service is the defacto standard for container orchestration



Azure Container Registry lets you store and manage container images in a central registry

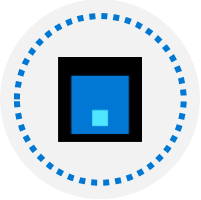


Azure Service Fabric to build and operate always-on, scalable, distributed apps

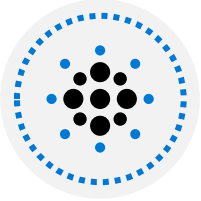


Azure App Service provides a managed service for both Windows and Linux based web applications

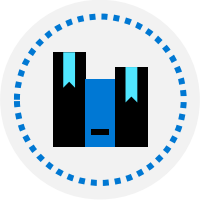
Docker container registries



(Public) Docker Hub <https://hub.docker.com>



(Public) Red Hat Container Catalog <https://access.redhat.com/containers>



(Public) Microsoft Container Registry <https://mcr.microsoft.com>



(Private) Create your own registry (Azure Container Registry makes this easy)

Demonstration: Create an Azure Container Registry

Container Registry

Microsoft

Azure Container Registry is a private registry for hosting container images. Using the Azure Container Registry, you can store Docker-formatted images for all types of container deployments. Azure Container Registry integrates well with orchestrators hosted in Azure Container Service, including Docker Swarm, DC/OS, and Kubernetes. Users can benefit from using familiar tooling capable of working with the open-source Docker Registry v2.

Use Azure Container Registry to:

- Store and manage container images across all types of Azure deployments
- Use familiar, open-source Docker command line interface (CLI) tools
- Keep container images near deployments to reduce latency and costs
- Simplify registry access management with Azure Active Directory
- Maintain Windows and Linux container images in a single Docker registry

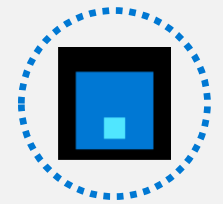
Dockerfile core concepts

```
FROM ubuntu
LABEL maintainer="johndoe@contoso.com"
ADD appsetup /
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
CMD ["echo", "Hello World from within the container"]
```



Dockerfiles are text files that contain the commands needed by **docker build** to assemble an image

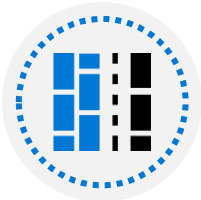
Lesson 3: Implementing Docker multi-stage builds



Multiple stage builds



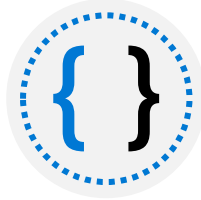
Keep the image size as small as possible



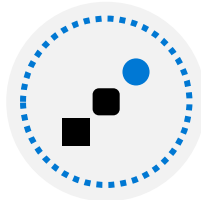
Layers are additional instructions added to the Dockerfile



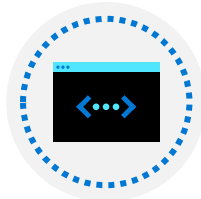
Multi-stage builds helps optimize the files, improves their readability, and makes them easier to maintain



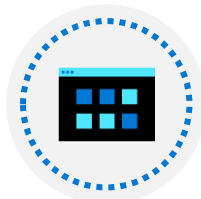
Each FROM instruction starts a new stage



The stages are numbered in order, starting with stage 0



Stages are named using an AS clause



Naming stages lets you build them separately

Multi-stage Dockerfiles

Multi-stage builds give the benefits of the builder pattern without the hassle of maintaining separate files

The code progresses from one stage to the next

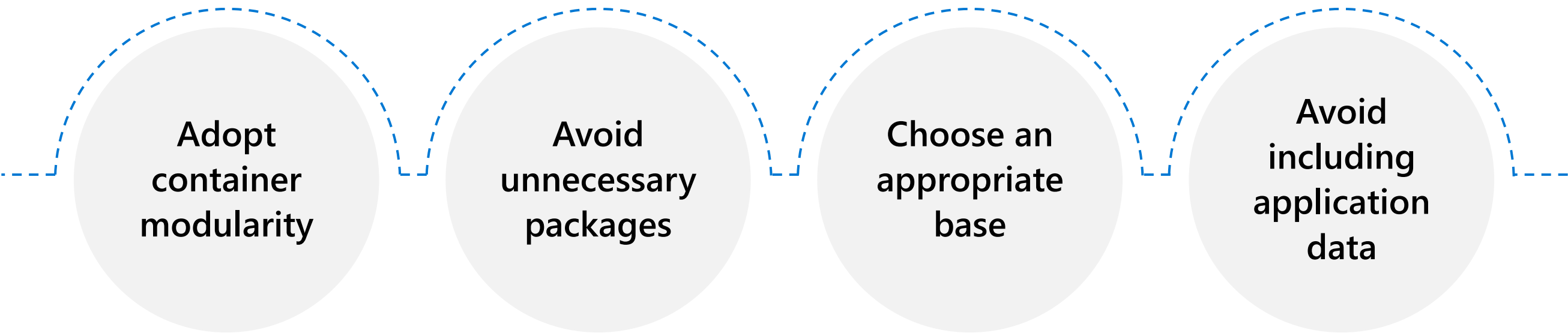
```
FROM microsoft/aspnetcore:2.0 AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/aspnetcore-build:2.0 AS builder
WORKDIR /src
COPY *.sln ./
COPY Web/Web.csproj Web/
RUN dotnet restore
COPY . .
WORKDIR /src/Web
RUN dotnet build -c Release -o /app

FROM builder AS publish
RUN dotnet publish -c Release -o /app

FROM base AS production
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "Web.dll"]
```

Considerations for multiple stage builds



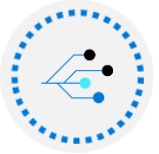
**Adopt
container
modularity**

**Avoid
unnecessary
packages**

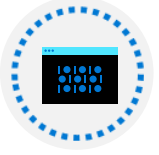
**Choose an
appropriate
base**

**Avoid
including
application
data**

Example builder pattern



Derive from a .NET base image with the whole runtime/SDK (Dockerfile.build)



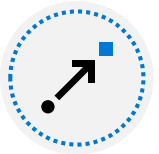
Add source code



Produce a statically-linked binary



Copy the static binary from the image to the host (docker create, docker cp)



Derive from SCRATCH or some other light-weight image (Dockerfile)



Add the binary back in



Push a tiny image to the Docker Hub

Multiple projects and solutions

Each project has its own multi-stage Dockerfile

Shared component projects do not have Dockerfiles

Each Dockerfile assumes its context is the solution directory

There's a docker-compose.yml in the root of the solution

We can now build the solution with a single docker command: **docker-compose build**

```
Multi.sln
docker-compose.yml
[Api]
    Dockerfile
[Web]
    Dockerfile
```

Demonstration: Add Docker support to an existing application

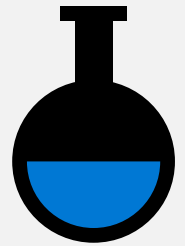
```
FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 6636
EXPOSE 44320

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY ["WebApplication1/WebApplication1.csproj", "WebApplication1/"]
...
RUN dotnet build "WebApplication1.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

Lesson 04: Lab



Deploying Docker containers to Azure App Service web apps



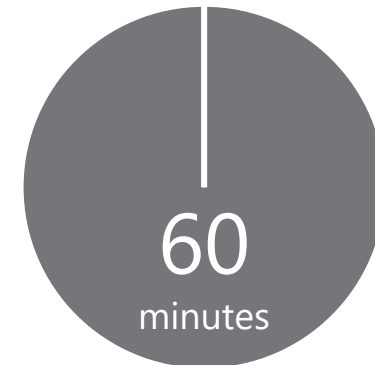
Lab overview:

In this lab, you will learn how to use an Azure DevOps CI/CD pipeline to build a custom Docker image, push it to Azure Container Registry, and deploy it as a container to Azure App Service.

Objectives:

- Build a custom Docker image by using a Microsoft hosted Linux agent
- Push an image to Azure Container Registry
- Deploy a Docker image as a container to Azure App Service by using Azure DevOps

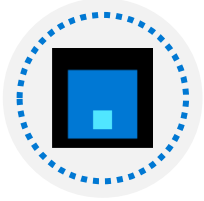
Duration:



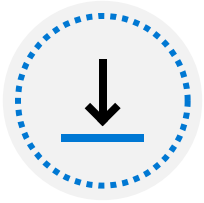
Lesson 05: Module review and takeaways



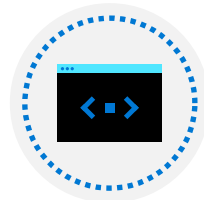
What did you learn ?



Implement a container strategy including how containers are different from virtual machines and how microservices use containers



Implement containers using Docker



Implement Docker Multi-Stage Builds

Module review questions

1

You are reviewing an existing Dockerfile. How would you know if it's a multi-stage Dockerfile?

2

You are designing a multi-stage Dockerfile. How can one stage refer to another stage within the Dockerfile?

3

What is the line continuation character in Dockerfiles?

4

When the Open Container Initiative defined a standard container image file format, which format did they choose as a starting point?