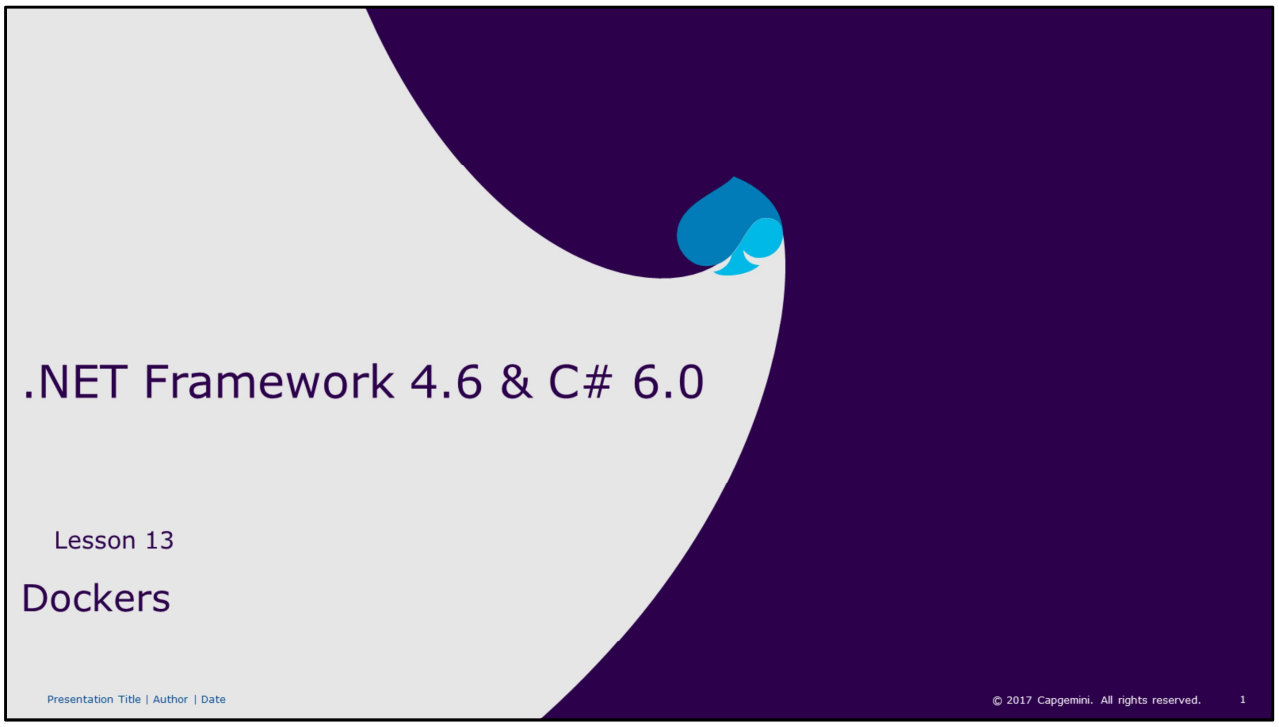

.NET Framework 4.6 & C# 6.0
Lesson 13
Dockers
Presentation Title | Author | Date
© 2017 Capgemini. All rights reserved.
1

Lesson Objectives

- What is a Docker?
- Choosing Between .NET Core and .NET Framework for Docker Containers
- When to choose .NET Core for Docker containers
- When to choose .NET Framework for Docker containers
- Decision table: .NET frameworks to use for Docker
- Docker Architecture
- Docker Engine
- Components of Docker
- Dockers Application Lifecycle
- Where to use Dockers?
- Scenarios for .NET Applications
- Visual Studio Tool for Docker

What is a Docker?

The Docker platform is the only container platform to build, secure and manage the widest array of applications from development to production both on premises and in the clouds

Below are Features of Dockers

- Openess
- Portability
- Independence
- Security
- Cost Savings
- Agility

Choosing Between .NET Core and .NET Framework for Docker Containers

There are two supported implementations for building server-side containerized Docker applications with .NET.

- .NET Framework
- .NET Core

- They share many .NET Standard components, and you can share code across the two.
- You should use .NET Core, with Linux or Windows Containers, for your containerized Docker server application when:
  - You have cross-platform needs. For example, you want to use both Linux and Windows Containers
  - Your application architecture is based on microservices.
  - You need to start containers fast and want a small footprint per container to achieve better density or more containers per hardware unit in order to lower your costs.

When you create new containerized .NET applications, you should consider .NET Core as the default choice. It has many benefits and fits best with the containers philosophy and style of working.
An additional benefit of using .NET Core is that you can run side by side .NET versions for applications within the same machine.

Choosing Between .NET Core and .NET Framework for Docker Containers

- You should use .NET Framework, with Windows Containers, for your containerized Docker server application when:
  - Your application currently uses .NET Framework and has strong dependencies on Windows.
  - You need to use Windows APIs that are not supported by .NET Core.
  - You need to use third-party .NET libraries or NuGet packages that are not available for .NET Core.

- Using .NET Framework on Docker can improve your deployment experiences by minimizing deployment issues.
- This "lift and shift" scenario is important for containerizing legacy applications that were originally developed with the traditional .NET Framework, like ASP.NET WebForms, MVC web apps or WCF (Windows Communication Foundation) services.

When to choose .NET Core for Docker containers

- Developing and deploying cross platform
- Using containers for new ("green-field") projects
- Creating and deploying microservices on containers
- Deploying high density in scalable systems

When to choose .NET Framework for Docker containers

- Migrating existing applications directly to a Windows Server container
- Using third-party .NET libraries or NuGet packages not available for .NET Core
- Using.NET technologies not available for .NET Core
  - ASP.NET Web Forms
  - WCF services
  - Workflow-related services. Windows Workflow Foundation (WF), Workflow Services (WCF + WF in a single service), and WCF Data Services (formerly known as ADO.NET Data Services) are only available on .NET Framework.
- Using a platform or API that does not support .NET Core

Using.NET technologies not available for .NET Core
ASP.NET Web Forms. This technology is only available on .NET Framework. Currently there are no plans to bring ASP.NET Web Forms to .NET Core.
WCF services. Even when a WCF-Client library is available to consume WCF services from .NET Core. as of mid-2017, the WCF server implementation is only available on .NET Framework. This scenario might be considered for future releases of .NET Core.
Workflow-related services. Windows Workflow Foundation (WF), Workflow Services (WCF + WF in a single service), and WCF Data Services (formerly known as ADO.NET Data Services) are only available on .NET Framework. There are currently no plans to bring them to .NET Core.

Decision table: .NET frameworks to use for Docker

- The following summarizes whether to use .NET Framework or .NET Core, and Windows or Linux containers. Remember that for Linux containers, you need Linux-based Docker hosts (VMs or servers) and that for Windows Containers you need Windows Server based Docker hosts (VMs or servers).

- Your application architecture choice is **Microservices on containers**.
  - Your .NET implementation choice should be .NET Core.
  - Your container platform choice can be either Linux containers or Windows containers.

- Your application architecture choice is a **Monolithic application**.
  - Your .NET implementation choice can be either .NET Core or .NET Framework.
  - If you have chosen .NET Core, your container platform choice can be either Linux containers or Windows containers.

- Your application is a **New container-based development ("green-field").**
  - Your .NET implementation choice should be .NET Core.
  - Your container platform choice can be either Linux containers or Windows containers.

Decision table: .NET frameworks to use for Docker

- Your application is a **Windows Server legacy app ("brown-field") migration to containers**.
  - Your .NET implementation choice is .NET Framework based on framework dependency.
  - Your container platform choice must be Windows containers because of the .NET Framework dependency.
- Your application's design goal is **Best-in-class performance and scalability**.
  - Your .NET implementation choice should be .NET Core.
  - Your container platform choice can be either Linux containers or Windows containers.
- You built your application using **ASP.NET Core.**
  - Your .NET implementation choice should be .NET Core.
  - You can use the .NET Framework implementation, if you have other framework dependencies.
  - If you have chosen .NET Core, your container platform choice can be either Linux containers or Windows containers.
  - If you have chosen .NET Framework, your container platform choice must be Windows containers.
- You built your application using **ASP.NET 4 (MVC 5, Web API 2, and Web Forms).**
  - Your .NET implementation choice is .NET Framework based on framework dependency.
  - Your container platform choice must be Windows containers because of the .NET Framework dependency.

Decision table: .NET frameworks to use for Docker

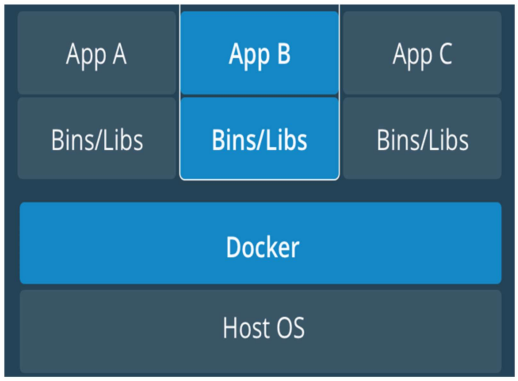
- Your application uses **SignalR services**.
  - Your .NET implementation choice can be .NET Framework, or .NET Core 2.1 (when released) or later.
  - Your container platform choice must be Windows containers if you chose the SignalR implementation in.NET Framework.
  - Your container platform choice can be either Linux containers or Windows containers if you chose the SignalR implementation in .NET Core 2.1 or later (when released).
  - When **SignalR services** run on .NET Core, you can use Linux containers or Windows Containers.
- Your application uses **WCF, WF, and other legacy frameworks**.
  - Your .NET implementation choice is .NET Framework, or .NET Core (in the roadmap for a future release).
  - Your container platform choice must be Windows containers because of the .NET Framework dependency.
- Your application involves **Consumption of Azure services**.
  - Your .NET implementation choice is .NET Framework, or .NET Core (eventually all Azure services will provide client SDKs for .NET Core).
  - Your container platform choice must be Windows containers if you use .NET Framework client APIs.
  - If you have chosen .NET Core, your container platform choice can be either Linux containers or Windows containers.
  - If you use client APIs available for .NET Core, you can also choose between Linux containers and Windows containers.

Docker – Image,Container,Compose

- Docker provides tooling and a platform to manage the lifecycle of your containers:
- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you’re ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

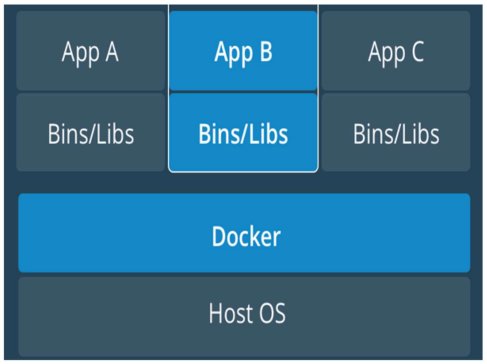
Docker – Image,Container,Compose

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production..



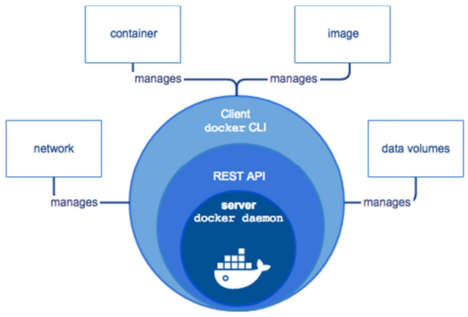
Docker – Image,Container,Compose

- An **image** is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.
- A **container** is a runtime instance of an image—what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.
- **Compose** is used to define applications using multiple Docker containers.



Docker – Engine

- Docker Engine is a client-server application with these major components:
- A server which is a type of long-running program called a daemon process (the dockerd command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.
The daemon creates and manages Docker *objects*, such as images, containers, networks, and volumes.

Components of Docker

Docker for Mac

- It allows one to run Docker containers on the Mac OS.

Docker for Linux

- It allows one to run Docker containers on the Linux OS.

Docker for Windows

- It allows one to run Docker containers on the Windows OS.

Docker Engine

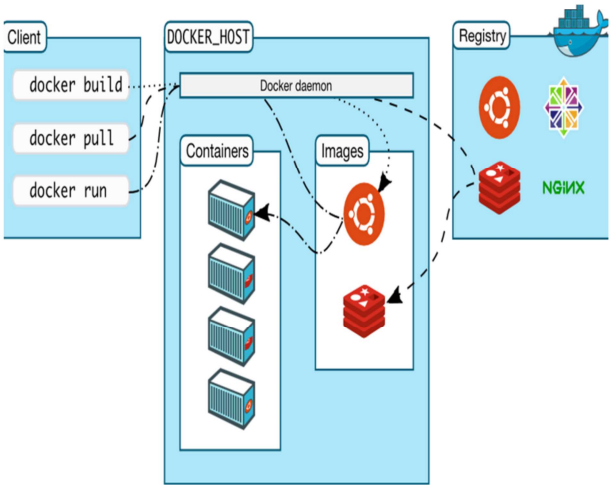
- It is used for building Docker images and creating Docker containers

Docker Hub

- This is the registry which is used to host various Docker images.

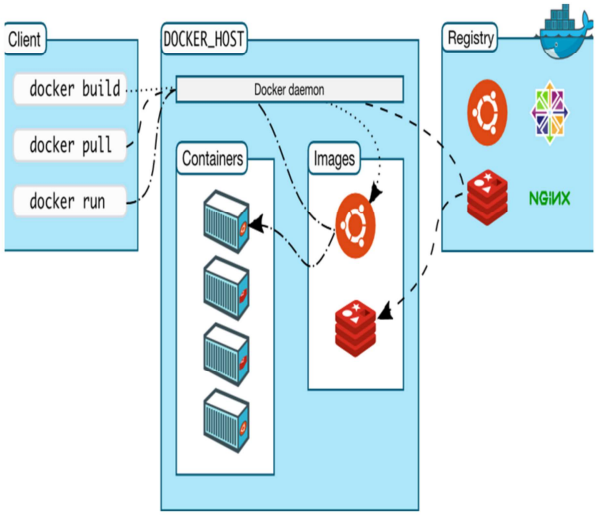
Dockers Architecture

- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface



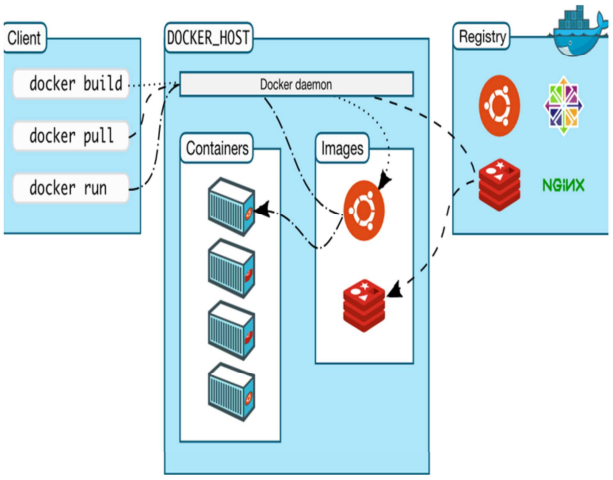
Dockers Architecture

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes
- The Docker client (docker) is the primary way that many Docker users interact with Docker
- A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry



Dockers Architecture

- Docker Objects
  - Images :An image is a read-only template with instructions for creating a Docker container.
  - Container:A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.
  - Services:Services allow you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers. Each member of a swarm is a Docker daemon, and the daemons all communicate using the Docker API.



You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time

Dockers Application Life Cycle

- The life cycle of containerized applications is a journey that begins with the developer.
- The developer chooses to implement containers and Docker because it eliminates frictions in deployments and IT operations, which ultimately helps everyone to be more agile, more productive end-to-end, and faster

Where to use Dockers?

Consider the following example scenario:

Your developers write code locally and share their work with their colleagues using Docker containers.

They use Docker to push their applications into a test environment and execute automated and manual tests.

When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.

When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment..

Scenarios for .NET Applications

Low friction install — You can try out .NET without installing anything on your machine. Just download a Docker image with .NET in it.

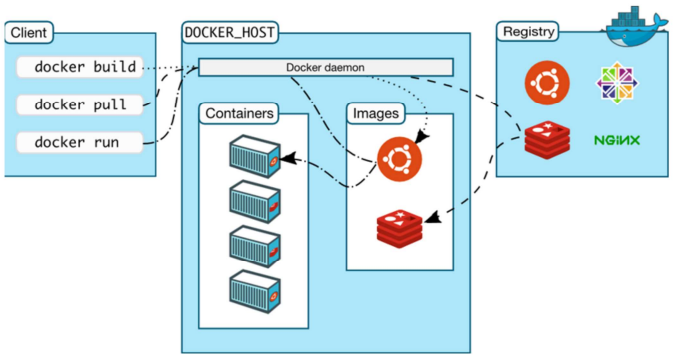
Develop in a container — You can develop in a consistent environment, making development and production environments very similar (avoiding issues like global state on developer machines).

Test in a container — You can test in a container, reducing failures due to incorrectly configured environments or other changes left behind from the last test.

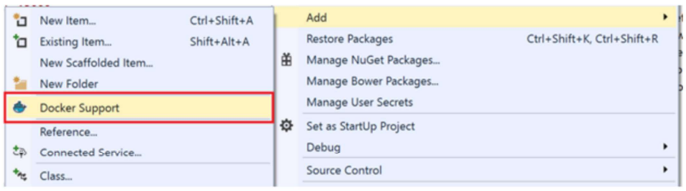
Scenarios for .NET Applications

Build in a container — You can build code in a container, avoiding the need to correctly configure shared build machines for multiple environments but instead move to a "BYOC" (bring your own container) approach.

Deployment in test, stage, production and other environments — You can deploy an image through all of your environments, reducing failures due to differences in configuration, typically only changing the behavior of the image via external configuration (for example, injected environment variables).

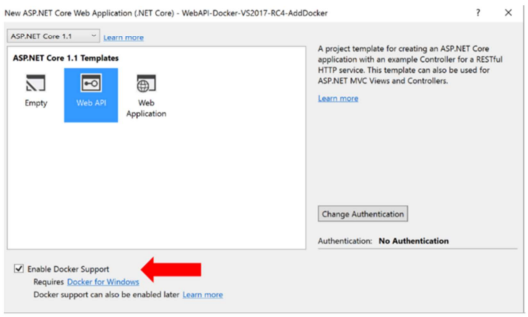


Visual Studio Tool for Docker

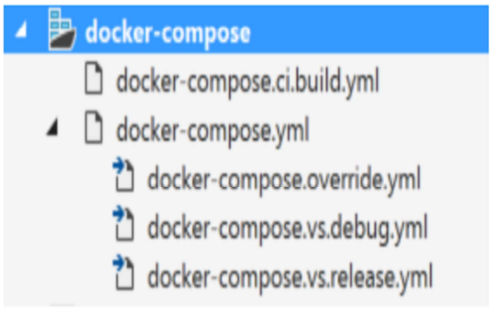


- The Visual Studio Tools for Docker provides a consistent way to develop and validate locally your Docker containers for Linux in a Linux Docker host or VM, or your Windows Containers directly on Windows.
- When you install the Visual Studio Tools for Docker, you get an easy integration into Docker.
- The tools will create a base Dockerfile and docker-compose.yml files that are now tied into your environment; so when you press F5, Visual Studio builds and runs the container using the Dockerfile and the debugging is now taking place in the running container!

Visual Studio Tool for Docker

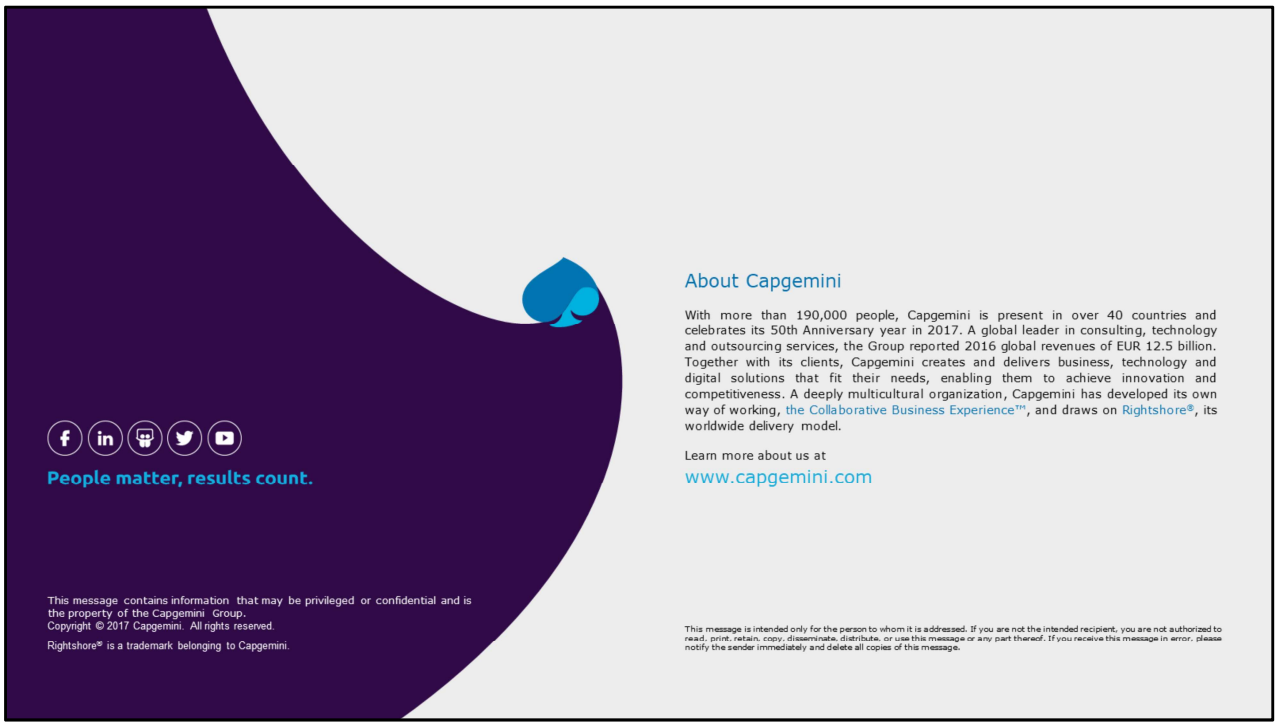


- Docker Support in Visual Studio 2017



- After you add Docker support to your solution in Visual Studio, you also will see a new node tree in Solution Explorer with the added docker-compose.yml files

You could deploy a multicontainer application by using a single docker-compose.yml file when you run docker-compose up; however, Visual Studio adds a group of them, so you can override values depending on the environment (development versus production) and the execution type (release versus debug).


People matter, results count.
This message contains information that may be privileged or confidential and is the property of the Capgemini Group.
Copyright © 2017 Capgemini. All rights reserved.
Rightshore® is a trademark belonging to Capgemini.
About Capgemini
With more than 190,000 people, Capgemini is present in over 40 countries and celebrates its 50th Anniversary year in 2017. A global leader in consulting, technology and outsourcing services, the Group reported 2016 global revenues of EUR 12.5 billion. Together with its clients, Capgemini creates and delivers business, technology and digital solutions that fit their needs, enabling them to achieve innovation and competitiveness. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.
Learn more about us at
www.capgemini.com
This message is intended only for the person to whom it is addressed. If you are not the intended recipient, you are not authorized to read, print, retain, copy, disseminate, distribute, or use this message or any part thereof. If you receive this message in error, please notify the sender immediately and delete all copies of this message.