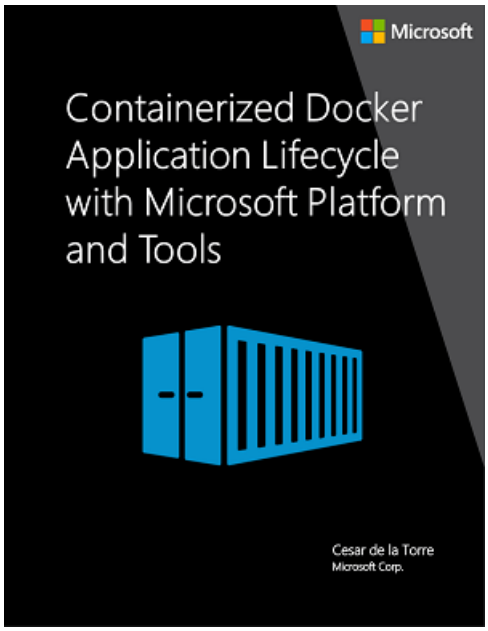
Contents

Ciclo de vida de aplicaciones de Docker en contenedor con la plataforma y las herramientas de Microsoft

06/03/2021 • 3 minutes to read • Edit Online



EDICIÓN v5.0: actualizada a ASP.NET Core 5.0

Consulte el registro de cambios para ver las modificaciones del libro y las colaboraciones para la comunidad.

En esta guía se incluye información general para desarrollar e implementar aplicaciones ASP.NET Core en contenedores con Docker, mediante la plataforma y las herramientas de Microsoft. En ella también se incluye una presentación de alto nivel de Azure DevOps, a fin de implementar canalizaciones de CI/CD, así como Azure Container Registry (ACR) y Azure Kubernetes Services (AKS) para la implementación.

Para obtener detalles de bajo nivel relacionados con el desarrollo, puede ver la guía Microservicios de .NET: Arquitectura para aplicaciones .NET en contenedor y su aplicación de referencia relacionada, eShopOnContainers.

Envíenos sus comentarios.

Hemos creado esta guía para ayudarle a entender la arquitectura de aplicaciones y microservicios en contenedor en .NET. La guía y la aplicación de referencia relacionada irán evolucionando, por lo que le agradecemos sus comentarios. Si tiene comentarios sobre cómo se puede mejorar esta guía, envíelos a https://aka.ms/ebookfeedback.

Créditos

Autor:

Cesar de la Torre, administrador de programas sénior del equipo de producto de .NET, Microsoft Corp.

Editor de adquisiciones:

Janine Patrick

Editor de desarrollo:

Bob Russell, profesional de soluciones en Microsoft

Octal Publishing, Inc.

Producción editorial:

Dianne Russell

Octal Publishing, Inc.

Revisor:

Bob Russell, profesional de soluciones en Microsoft

Participantes y revisores:

Nish Anil, director de administración de programas, equipo de .NET, Microsoft

Miguel Veloso, ingeniero de desarrollo de software en Plain Concepts

Sumit Ghosh, asesor principal en Neudesic

Copyright

PUBLICADO POR

Equipos de producto de la División de desarrolladores de Microsoft, .NET y Visual Studio

División de Microsoft Corporation

One Microsoft Way

Redmond, Washington 98052-6399

SIGUIENTE

Introducción a Containers y Docker

02/11/2020 • 3 minutes to read • Edit Online

La creación de contenedores es un enfoque de desarrollo de software en el que una aplicación o un servicio, sus dependencias y su configuración (extraídos como archivos de manifiesto de implementación) se empaquetan como una imagen de contenedor. Puede probar la aplicación en contenedor como una unidad e implementarla como una instancia de imagen de contenedor en el sistema operativo host.

Del mismo modo que los contenedores de mercancías permiten su transporte por barco, tren o camión independientemente de la carga de su interior, los contenedores de software actúan como una unidad estándar de implementación de software que puede contener diferentes dependencias y código. De esta manera, la inclusión del software en contenedor permite a los desarrolladores y los profesionales de TI implementarlo en entornos con pocas modificaciones o ninguna en absoluto.

Los contenedores también aíslan las aplicaciones entre sí en un sistema operativo compartido. Las aplicaciones en contenedor se ejecutan sobre un host de contenedor que a su vez se ejecuta en el sistema operativo (Linux o Windows). Por lo tanto, los contenedores tienen una superficie mucho menor que las imágenes de máquina virtual (VM).

Cada contenedor puede ejecutar una aplicación web o un servicio al completo, como se muestra en la figura 1-1. En este ejemplo, el host de Docker es un host de contenedor, y App 1, App 2, Svc 1 y Svc 2 son aplicaciones o servicios en contenedor.

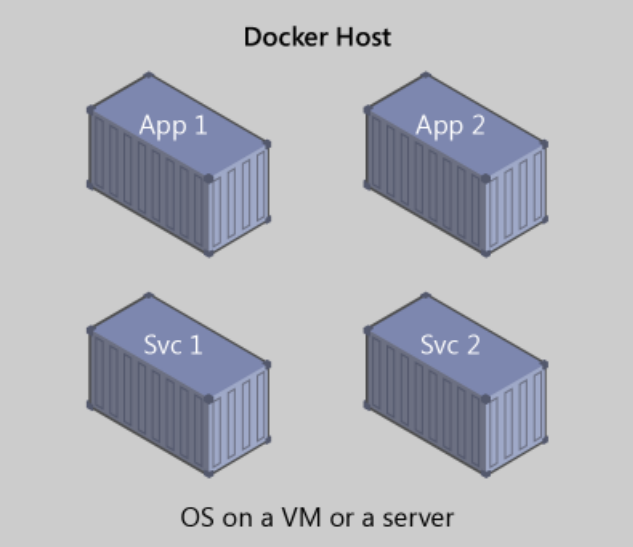


Figura 1-1. Varios contenedores ejecutándose en un host de contenedor.

Otra ventaja derivada de la inclusión en contenedores es la escalabilidad. La creación de contenedores para tareas a corto plazo permite escalar horizontalmente con gran rapidez. Desde el punto de vista de la aplicación, la creación de instancias de una imagen (la creación de un contenedor) es similar a la creación de instancias de un proceso como un servicio o una aplicación web. Pero con fines de confiabilidad, cuando ejecute varias instancias de la misma imagen en varios servidores host, seguramente le interesará que cada contenedor (instancia de imagen) se ejecute en un servidor host o máquina virtual diferente en dominios de error distintos.

En resumen, los contenedores ofrecen las ventajas del aislamiento, la portabilidad, la agilidad, la escalabilidad y el control a lo largo de todo el flujo de trabajo del ciclo de vida de la aplicación. La ventaja más importante es el aislamiento del entorno que se proporciona entre el desarrollo y las operaciones.

ANTERIOR SIGUIENTE

¿Qué es Docker?

02/11/2020 • 10 minutes to read • Edit Online

Docker es un proyecto de código abierto para automatizar la implementación de aplicaciones como contenedores portátiles y autosuficientes que se pueden ejecutar en la nube o localmente. Docker es también una empresa que promueve e impulsa esta tecnología, en colaboración con proveedores de la nube, Linux y Windows, incluido Microsoft.

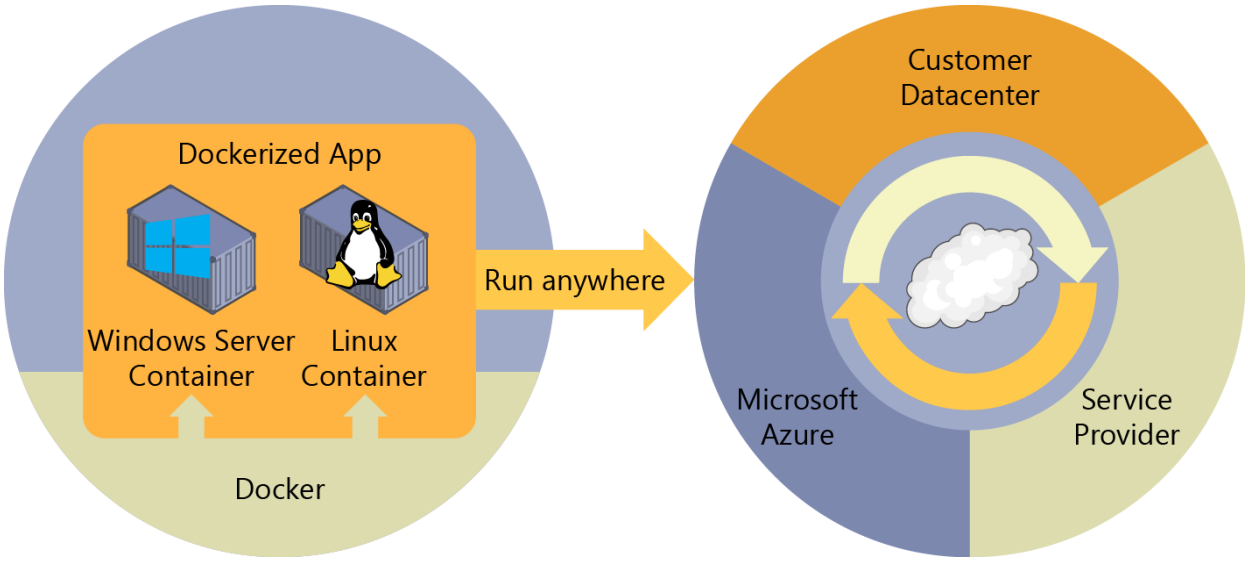


Figura 1-2. Docker implementa contenedores en todas las capas de la nube híbrida

Tal y como se muestra en el diagrama anterior, los contenedores de Docker se pueden ejecutar en cualquier lugar, localmente en el centro de recursos de cliente, en un proveedor de servicios externo o en la nube, en Azure. Los contenedores de imágenes de Docker también se pueden ejecutar de forma nativa en Linux y Windows. Sin embargo, las imágenes de Windows solo pueden ejecutarse en hosts de Windows y las imágenes de Linux pueden ejecutarse en hosts de Linux y hosts de Windows (con una máquina virtual Linux de Hyper-V, hasta el momento), donde host significa un servidor o una máquina virtual.

Los desarrolladores pueden usar entornos de desarrollo en Windows, Linux o macOS. En el equipo de desarrollo, el desarrollador ejecuta un host de Docker en que se implementan imágenes de Docker, incluidas la aplicación y sus dependencias. Los desarrolladores que trabajan en Linux o en Mac usan un host de Docker basado en Linux y solo pueden crear imágenes para contenedores de Linux. (Los desarrolladores que trabajan en Mac pueden editar código o ejecutar la interfaz de la línea de comandos, o CLI, de Docker en macOS, pero en el momento de redactar este artículo, los contenedores no se ejecutan directamente en macOS). Los desarrolladores que trabajan en Windows pueden crear imágenes para contenedores de Windows o Linux.

Para hospedar contenedores en entornos de desarrollo y proporcionar herramientas de desarrollador adicionales, Docker entrega Docker Community Edition (CE) para Windows o macOS. Estos productos instalan la máquina virtual necesaria (el host de Docker) para hospedar los contenedores. Docker también pone a disposición Docker Enterprise Edition (EE), que está diseñado para el desarrollo empresarial y se usa en los equipos de TI que crean, envían y ejecutan aplicaciones críticas para la empresa en producción.

Para ejecutar contenedores de Windows, hay dos tipos de tiempos de ejecución:

- Los **contenedores de Windows Server** ofrecen aislamiento de aplicaciones a través de tecnología de aislamiento de proceso y de espacio de nombres. Un contenedor de Windows Server comparte el kernel con el host de contenedor y con todos los contenedores que se ejecutan en el host.

- Los **contenedores de Hyper-V** amplían el aislamiento que ofrecen los contenedores de Windows Server mediante la ejecución de cada contenedor en una máquina virtual altamente optimizada. En esta configuración, el kernel del host del contenedor no se comparte con los contenedores de Hyper-V, lo que proporciona un mejor aislamiento.

Las imágenes de estos contenedores se crean y funcionan de la misma manera. La diferencia radica en cómo se crea el contenedor desde la imagen ejecutando un contenedor de Hyper-V que requiere un parámetro adicional. Para más información, vea Contenedores de Hyper-V.

Comparación de los contenedores de Docker con las máquinas virtuales

En la figura 1-3 se muestra una comparación entre las máquinas virtuales y los contenedores de Docker.

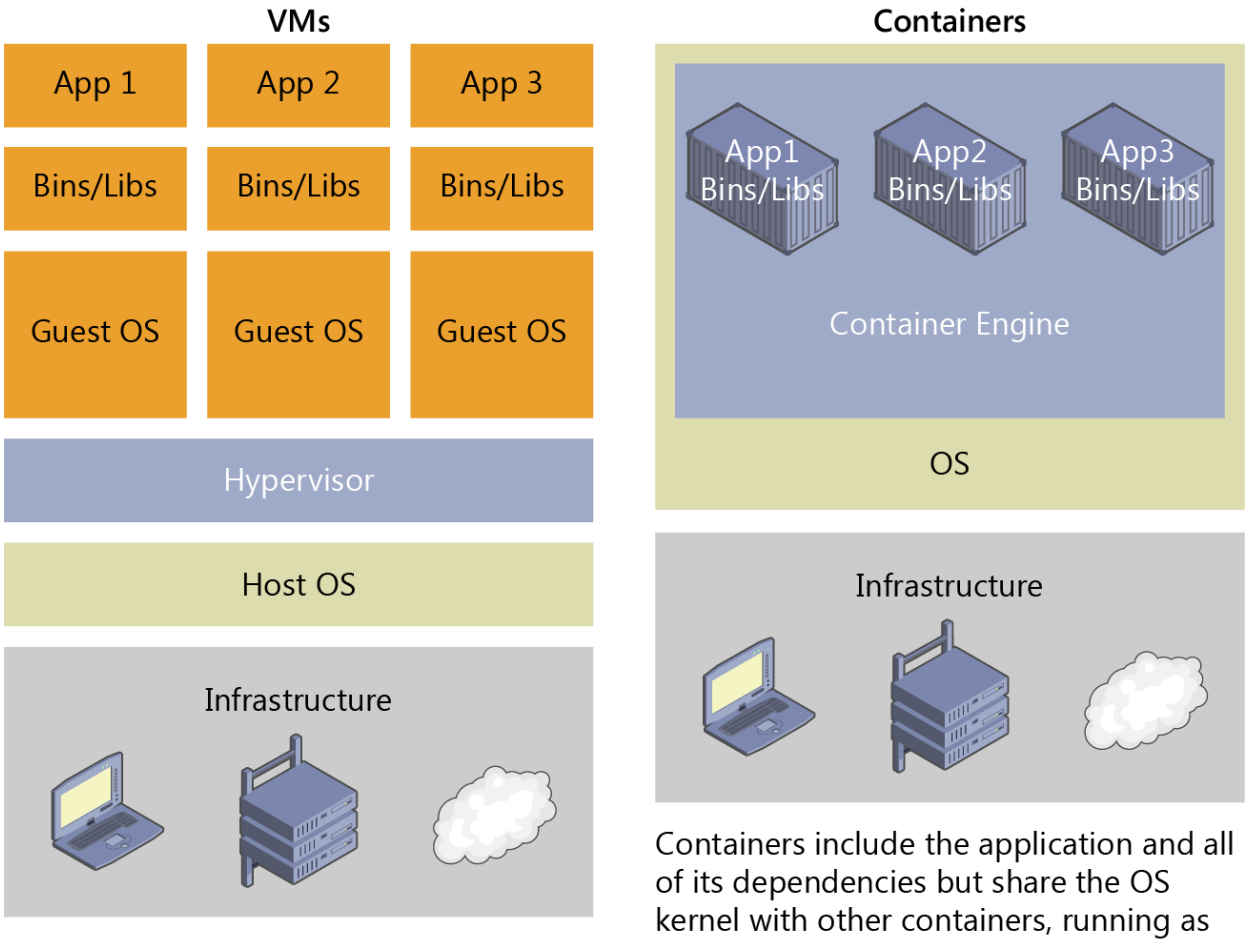


Figura 1-3. Comparación de las máquinas virtuales tradicionales con los contenedores de Docker

Tal y como se muestra en el diagrama anterior, en el caso de las máquinas virtuales, hay tres capas base en el servidor host. De abajo hacia arriba: infraestructura, sistema operativo host y un hipervisor. Encima de todo eso, cada máquina virtual tiene su propio sistema operativo y todas las bibliotecas necesarias. Por otro lado, para Docker, el servidor host solo tiene la infraestructura y el sistema operativo. Además, el motor de contenedor mantiene los contenedores aislados, pero les permite compartir los servicios del sistema operativo de base única.

Dado que los contenedores requieren muchos menos recursos (por ejemplo, no necesitan un sistema operativo completo), se inician rápidamente y son fáciles de implementar. Esto permite tener una mayor densidad, lo que

significa que se pueden ejecutar más servicios en la misma unidad de hardware, reduciendo así los costos.

Como efecto secundario de la ejecución en el mismo kernel, obtiene menos aislamiento que las máquinas virtuales.

El objetivo principal de una imagen consiste en garantizar el mismo entorno (dependencias) entre las distintas implementaciones. Esto significa que puede depurarlo en su equipo y, después, implementarlo en otra máquina con el mismo entorno garantizado.

Una imagen de contenedor es una manera de empaquetar una aplicación o un servicio e implementarlo de forma confiable y reproducible. Podría decir que Docker no solo es una tecnología, sino también una filosofía y un proceso.

Al usar Docker, no escuchará a los desarrolladores decir "Si funciona en mi máquina, ¿por qué no en producción?". Pueden decir simplemente "Se ejecuta en Docker", porque la aplicación de Docker empaquetada puede ejecutarse en cualquier entorno de Docker compatible, y se ejecuta de la forma prevista en todos los destinos de implementación (como desarrollo, control de calidad, almacenamiento provisional y producción).

Una analogía simple

Quizás una analogía simple puede ayudar a entender el concepto básico de Docker.

Vamos a remontarnos a la década de 1950 por un momento. No había ningún procesador de texto y las fotocopiadoras se usaban en todas partes (o casi).

Imagine que es responsable de enviar lotes de cartas, según proceda, para enviarlas por correo a los clientes en papel y sobres reales, que se entregarán físicamente en la dirección postal de cada cliente (entonces no existía el correo electrónico).

En algún momento, se da cuenta de que las cartas no son más que una composición de un conjunto grande de párrafos, que se eligen y ordenan según proceda, según el propósito de la carga, por lo que diseña un sistema para emitir cartas rápidamente, esperando conseguir una increíble mejora.

El sistema es simple:

1. Comience con un conjunto de hojas transparentes que contienen un párrafo.
2. Para emitir un conjunto de cartas, elija las hojas con los párrafos necesarios, apílelas y alinéelas para que queden y se lean bien.
3. Por último, colóquelas en la fotocopiadora y presione inicio para producir tantas cartas como sean necesarias.

Por tanto, para simplificar, esa es la idea principal de Docker.

En Docker, cada capa es el conjunto resultante de los cambios que se producen en el sistema de archivos después de ejecutar un comando, como instalar un programa.

Por lo tanto, al "Examinar" el sistema de archivos después de que se ha copiado la capa, verá todos los archivos, incluida la capa cuando se instaló el programa.

Puede pensar en una imagen como un disco duro de solo lectura auxiliar listo para instalarse en un "equipo" donde el sistema operativo ya está instalado.

De forma similar, puede pensar en un contenedor como el "equipo" con el disco duro de imagen instalado. El contenedor, como un equipo, se puede apagar o desactivar.

Terminología de Docker

06/03/2021 • 11 minutes to read • Edit Online

En esta sección se enumeran los términos y las definiciones que debe conocer antes de profundizar en el uso de Docker. Para consultar más definiciones, lea el amplio glosario que Docker proporciona.

Imagen de contenedor: un paquete con todas las dependencias y la información necesarias para crear un contenedor. Una imagen incluye todas las dependencias (por ejemplo, los marcos), así como la configuración de implementación y ejecución que usará el runtime de un contenedor. Normalmente, una imagen se deriva de varias imágenes base que son capas que se apilan unas encima de otras para formar el sistema de archivos del contenedor. Una vez que se crea una imagen, esta es inmutable.

Dockerfile: archivo de texto que contiene instrucciones sobre cómo compilar una imagen de Docker. Es como un script por lotes; la primera línea indica la imagen base con la que se comienza y, después, deben seguirse las instrucciones para instalar programas necesarios, copiar archivos, etc., hasta obtener el entorno de trabajo que se necesita.

Compilación: la acción de crear una imagen de contenedor basada en la información y el contexto que proporciona su Dockerfile, así como archivos adicionales en la carpeta en que se crea la imagen. Puede compilar imágenes con el siguiente comando de Docker:

```
docker build
```

Contenedor: una instancia de una imagen de Docker. Un contenedor representa la ejecución de una sola aplicación, proceso o servicio. Está formado por el contenido de una imagen de Docker, un entorno de ejecución y un conjunto estándar de instrucciones. Al escalar un servicio, crea varias instancias de un contenedor a partir de la misma imagen. O bien, un proceso por lotes puede crear varios contenedores a partir de la misma imagen y pasar parámetros diferentes a cada instancia.

Volúmenes: ofrece un sistema de archivos grabable que el contenedor puede usar. Puesto que las imágenes son de solo lectura pero la mayoría de los programas necesitan escribir en el sistema de archivos, los volúmenes agregan una capa grabable, encima de la imagen de contenedor, por lo que los programas tienen acceso a un sistema de archivos grabable. El programa no sabe que está accediendo a un sistema de archivos por capas, que no es más que el sistema de archivos habitual. Los volúmenes residen en el sistema host y los administra Docker.

Etiqueta: una marca o una etiqueta que se puede aplicar a las imágenes para que se puedan identificar diferentes imágenes o versiones de la misma imagen (según el número de versión o el entorno de destino).

Compilación de varias fases: es una característica, desde Docker 17.05 o versiones posteriores, que ayuda a reducir el tamaño de las imágenes finales. Por ejemplo, se puede usar una imagen base grande que contenga el SDK para compilar y publicar y una imagen base pequeña solo de tiempo de ejecución para hospedar la aplicación.

Repositorio: una colección de imágenes de Docker relacionadas, etiquetadas con una etiqueta que indica la versión de la imagen. Algunos repositorios contienen varias variantes de una imagen específica, como una imagen que contiene SDK (más pesada), una imagen que solo contiene runtimes (más ligera), etcétera. Estas variantes se pueden marcar con etiquetas. Un solo repositorio puede contener variantes de plataforma, como una imagen de Linux y una imagen de Windows.

Registro: servicio que proporciona acceso a los repositorios. El registro predeterminado para la mayoría de las

imágenes públicas es Docker Hub (propiedad de Docker como una organización). Normalmente, un registro contiene repositorios procedentes de varios equipos. Las empresas suelen tener registros privados para almacenar y administrar imágenes que han creado. Azure Container Registry es otro ejemplo.

Imagen multiarquitectura: En el caso de la arquitectura múltiple, es una característica que simplifica la selección de la imagen adecuada, según la plataforma donde se ejecuta Docker. Por ejemplo, si un Dockerfile solicita una imagen base **FROM mcr.microsoft.com/dotnet/sdk:5.0** del Registro, en realidad obtendrá **5.0-nanoserver-20H2**, **5.0-nanoserver-2004** o **5.0-buster-slim**, según el sistema operativo en el que se ejecute Docker y la versión.

Docker Hub: registro público para cargar imágenes y trabajar con ellas. Docker Hub proporciona hospedaje de imágenes de Docker, registros públicos o privados, desencadenadores de compilación y enlaces web e integración con GitHub y Bitbucket.

Azure Container Registry: recurso público para trabajar con imágenes de Docker y sus componentes en Azure. Esto proporciona un registro cercano a las implementaciones en Azure que le proporciona control sobre el acceso, lo que le permite usar los grupos y los permisos de Azure Active Directory.

Docker Trusted Registry (DTR) : servicio del registro de Docker (ofrecido por Docker) que se puede instalar de forma local, por lo que se encuentra en el centro de datos y la red de la organización. Es ideal para imágenes privadas que deben administrarse dentro de la empresa. Docker Trusted Registry se incluye como parte del producto Docker Datacenter. Para más información, vea Docker Trusted Registry (DTR).

Docker Community Edition (CE) : herramientas de desarrollo para Windows y MacOS para compilar, ejecutar y probar contenedores localmente. Docker CE para Windows proporciona entornos de desarrollo para contenedores de Windows y Linux. El host de Docker de Linux en Windows se basa en una máquina virtual Hyper-V. El host para los contenedores de Windows se basa directamente en Windows. Docker CE para Mac se basa en el marco del hipervisor de Apple y el hipervisor xhyve, lo que proporciona una máquina virtual de host de Docker de Linux en Mac OS X. Docker CE para Windows y para Mac sustituye a Docker Toolbox, que se basaba en Oracle VirtualBox.

Docker Enterprise Edition (EE) : versión a escala empresarial de las herramientas de Docker para el desarrollo de Linux y Windows.

Compose: herramienta de línea de comandos y formato de archivo YAML con metadatos para definir y ejecutar aplicaciones de varios contenedores. Primero se define una sola aplicación basada en varias imágenes con uno o más archivos .yml que pueden invalidar los valores según el entorno. Después de crear las definiciones, puede implementar toda la aplicación de varios contenedores con un solo comando (docker-compose up) que crea un contenedor por imagen en el host de Docker.

Clúster: colección de hosts de Docker que se expone como si fuera un solo host de Docker virtual, de manera que la aplicación se puede escalar a varias instancias de los servicios repartidos entre varios hosts del clúster. Los clústeres de Docker se pueden crear con Kubernetes, Azure Service Fabric, Docker Swarm y Mesosphere DC/OS.

Orquestador: herramienta que simplifica la administración de clústeres y hosts de Docker. Los orquestadores permiten administrar las imágenes, los contenedores y los hosts a través de una interfaz de la línea de comandos (CLI) o una interfaz gráfica de usuario. Puede administrar las redes de contenedor, las configuraciones, el equilibrio de carga, la detección de servicios, la alta disponibilidad, la configuración del host de Docker y muchas cosas más. Un orquestador se encarga de ejecutar, distribuir, escalar y reparar las cargas de trabajo a través de una colección de nodos. Normalmente, los productos del orquestador son los mismos que proporcionan infraestructura de clúster, como Kubernetes y Azure Service Fabric, entre otras ofertas del mercado.

Contenedores, imágenes y registros de Docker

02/11/2020 • 3 minutes to read • Edit Online

Cuando se utiliza Docker, se crea una aplicación o un servicio que se empaqueta con sus dependencias en una imagen de contenedor. Una imagen es una representación estática de la aplicación o el servicio y de su configuración y las dependencias.

Para ejecutar la aplicación o el servicio, se crea una instancia de la imagen de la aplicación para crear un contenedor, que se ejecutará en el host de Docker. Inicialmente, los contenedores se prueban en un entorno de desarrollo o un PC.

Las imágenes se almacenan en un registro, que sirve como biblioteca de imágenes. Se necesita un registro al implementar en orquestadores de producción. Docker mantiene un registro público a través de Docker Hub; otros proveedores ofrecen registros para distintas colecciones de imágenes, incluido Azure Container Registry. Como alternativa, las empresas pueden tener un registro privado local para sus propias imágenes de Docker.

En la figura 1-4 se muestra cómo se relacionan las imágenes y los registros de Docker con otros componentes. También se muestran las diversas ofertas de registro de los proveedores.

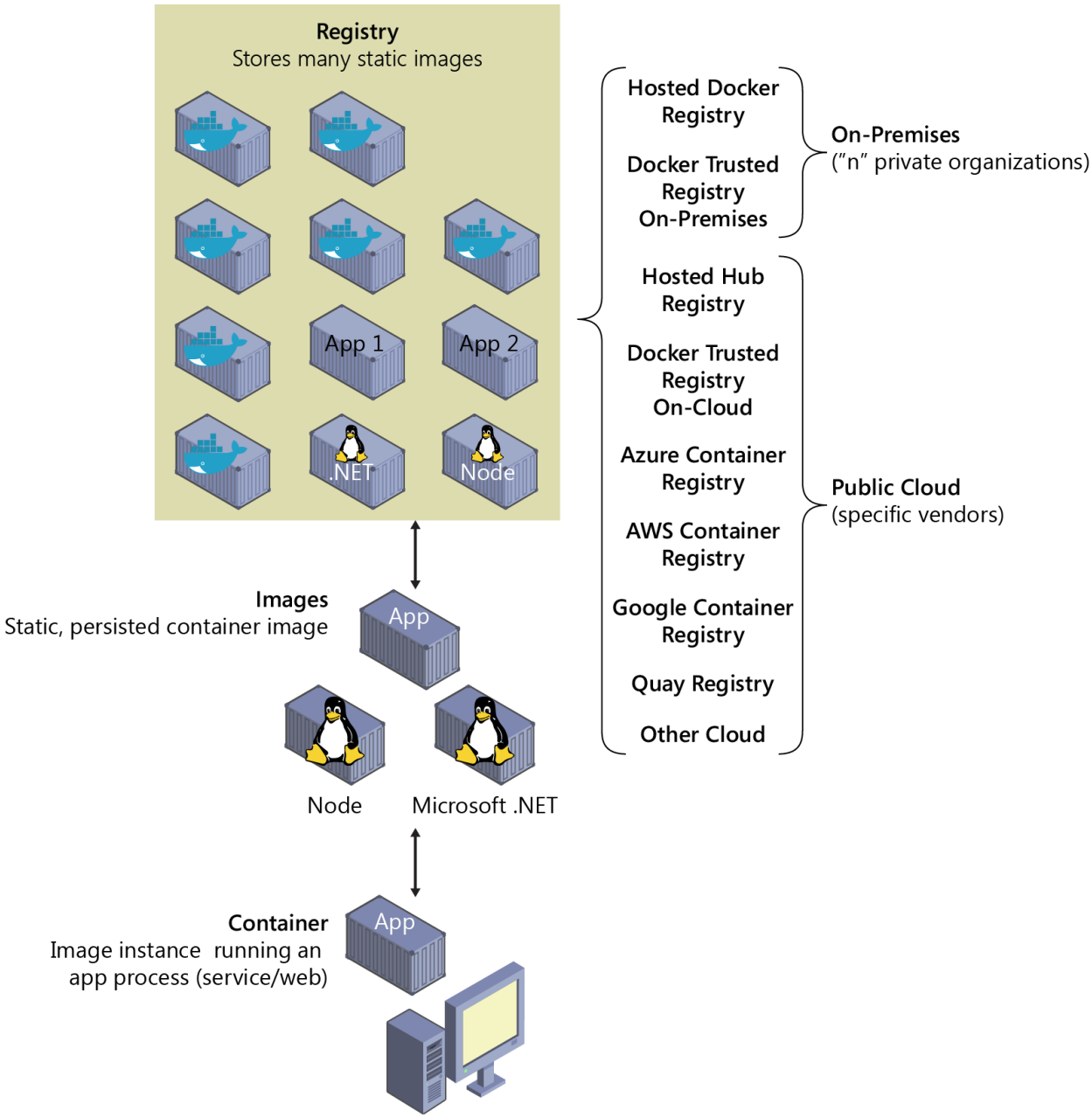


Figura 1-4. Taxonomía de términos de Docker y conceptos

el registro es como una estantería donde las imágenes se almacenan y están disponibles para extraerlas con el fin de compilar contenedores que ejecuten servicios o aplicaciones web. Hay registros de Docker privados a nivel local y en la nube pública. Docker Hub es que un registro público mantenido por Docker; junto con Docker Trusted Registry, una solución a nivel empresarial, Azure ofrece Azure Container Registry. AWS, Google y otros también tienen registros de contenedor.

Si coloca imágenes en un registro, puede almacenar bits de la aplicación estáticos e inmutables, con todas sus dependencias, a nivel de marco. Luego puede versionar e implementar imágenes en varios entornos y, por tanto, proporcionar una unidad de implementación coherente.

Los registros de imágenes privados, ya sean hospedados localmente o en la nube, se recomiendan cuando:

- Las imágenes no deben compartirse públicamente por motivos de confidencialidad.
- Quiere tener una latencia de red mínima entre las imágenes y el entorno de implementación elegido. Por ejemplo, si el entorno de producción es Azure, probablemente quiera almacenar las imágenes en Azure Container Registry, para que la latencia de red sea mínima. De forma similar, si el entorno de producción es local, puede tener un registro de confianza de Docker local disponible dentro de la misma red local.

ANTERIOR SIGUIENTE

El camino hacia las aplicaciones modernas basadas en contenedores

02/11/2020 • 2 minutes to read • Edit Online

Probablemente esté leyendo este libro porque planea el desarrollo de nuevas aplicaciones o evalúa el impacto de utilizar Docker, Containers y nuevos enfoques como Microservices en su empresa.

La adopción de nuevos paradigmas de desarrollo debe hacerse con cautela antes de iniciar un proyecto, para evaluar el impacto en sus equipos de desarrollo, su presupuesto o su infraestructura.

Microsoft ha estado trabajando en una guía avanzada, las aplicaciones de ejemplo y un conjunto de libros electrónicos, que pueden ayudarle a tomar una decisión informada y guiar a su equipo a través del desarrollo correcto, la implementación y las operaciones de sus nuevas aplicaciones.

Este libro forma parte de una serie de guías de Microsoft que tratan muchas de las necesidades y retos a los que se enfrentará durante el proceso de desarrollo de nuevas aplicaciones modernas basadas en contenedores.

Puede encontrar más libros electrónicos de Microsoft relacionados con los contenedores de Docker en la lista siguiente:

- **Microservicios de .NET. Arquitectura para aplicaciones .NET en contenedor**
  https://docs.microsoft.com/dotnet/architecture/microservices/
- **Modernización de las aplicaciones .NET existentes con la nube de Azure y los contenedores de Windows**
  https://docs.microsoft.com/dotnet/architecture/modernize-with-azure-containers/

ANTERIOR SIGUIENTE

Introducción al ciclo de vida de aplicaciones de Docker

02/11/2020 • 2 minutes to read • Edit Online

El ciclo de vida de aplicaciones en contenedor es una trayectoria que empieza por el desarrollador. El desarrollador decide implementar contenedores y Docker, ya que así se eliminan las fricciones en las implementaciones y las operaciones de TI, lo que, en última instancia, ayuda a todos los usuarios a ser más ágiles, más productivos de un extremo a otro y más rápidos.

ANTERIOR SIGUIENTE

Contenedores como base para la colaboración de DevOps

02/11/2020 • 15 minutes to read • Edit Online

Gracias a la naturaleza intrínseca de la tecnología de Docker y los contenedores, los desarrolladores pueden compartir el software y las dependencias fácilmente con los equipos de operaciones de TI y los entornos de producción, lo que pone fin a la típica excusa de "funciona en mi equipo". Los contenedores solucionan los conflictos de las aplicaciones entre distintos entornos. De manera indirecta, los contenedores y Docker acercan todavía más a los desarrolladores y los equipos de operaciones de TI, lo que les permite colaborar de forma eficaz. Al adoptar el flujo de trabajo de contenedor, muchos clientes tienen acceso a la continuidad de DevOps que tanto ansiaban y que antes debían implementar a través de una configuración más compleja de las canalizaciones de versión y compilación. Los contenedores simplifican las canalizaciones de compilación, prueba e implementación de DevOps.

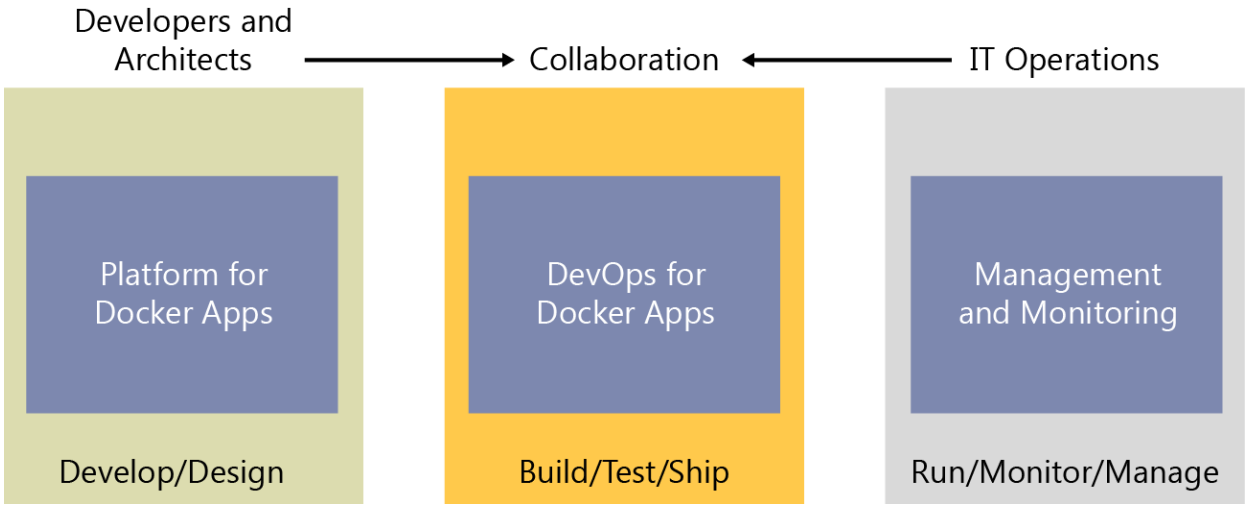


Figura 2-1. Cargas de trabajo principales por "roles" en el ciclo de vida de aplicaciones en contenedor de Docker.

Con los contenedores de Docker, los desarrolladores controlan lo que está dentro del contenedor (la aplicación y el servicio, así como las dependencias de los marcos y los componentes) y la manera en que los contenedores y los servicios se comportan juntos como una aplicación compuesta por una colección de servicios. Las interdependencias de los diversos contenedores se definen en un archivo `docker-compose.yml`, o lo que podría llamarse *manifiesto de implementación*. Mientras tanto, los equipos de operaciones de TI (profesionales de TI y administración) pueden centrarse en la gestión de los entornos de producción, la infraestructura, la escalabilidad, la supervisión y, en última instancia, la verificación de que las aplicaciones ofrecen un servicio adecuado a los usuarios finales, sin necesidad de conocer el contenido de los diversos contenedores. A esto se debe el término "contenedor," como analogía con los contenedores de transporte del mundo real. Los propietarios del contenido de un contenedor no necesitan preocuparse por cómo se enviará el contenedor y la empresa de transporte envía el contenedor desde el punto de origen al destino sin conocer el contenido ni preocuparse por él. De forma similar, los desarrolladores pueden crear y poseer el contenido de un contenedor de Docker sin necesidad de preocuparse por los mecanismos de "transporte".

En el bloque de la izquierda de la figura 2-1, los desarrolladores escriben y ejecutan código localmente en contenedores de Docker mediante Docker para Windows o Mac. Definen el entorno operativo para el código a través de un archivo Dockerfile que especifica el sistema operativo básico que se va a ejecutar, así como los pasos de compilación para compilar el código en una imagen de Docker. Los desarrolladores definen cómo interactuarán una o varias imágenes mediante el manifiesto de implementación del archivo `docker-compose.yml`

mencionado anteriormente. A medida que completan el desarrollo local, insertan el código de la aplicación y los archivos de configuración de Docker en el repositorio de código de su elección (es decir, el repositorio GIT).

El bloque de DevOps define las canalizaciones de compilación e integración continua (CI) mediante el archivo Dockerfile que se proporciona en el repositorio de código. El sistema de CI extrae las imágenes del contenedor base del registro de Docker seleccionado y compila las imágenes de Docker personalizadas para la aplicación. Después, las imágenes se validan y se insertan en el registro de Docker que se usa para las implementaciones en varios entornos.

En el bloque de la derecha, los equipos de operaciones administran la infraestructura y las aplicaciones implementadas en producción al mismo tiempo que supervisan el entorno y las aplicaciones, de modo que puedan proporcionar comentarios e información al equipo de desarrollo sobre la manera en que podría mejorarse la aplicación. Las aplicaciones de contenedor normalmente se ejecutan en producción mediante orquestadores de contenedores como Kubernetes. Asimismo, con frecuencia se emplean gráficos de Helm para configurar unidades de implementación, en lugar de archivos docker-compose.

Los dos equipos colaboran mediante una plataforma fundamental (contenedores de Docker) que proporciona una "separación de intereses" de forma contractual, al tiempo que mejora en gran medida la colaboración de los dos equipos en el ciclo de vida de la aplicación. Los desarrolladores poseen el contenido del contenedor, su entorno operativo y las interdependencias del contenedor, mientras que los equipos de operaciones toman las imágenes compiladas junto con el manifiesto y las ejecutan en su sistema de orquestación.

Desafíos del ciclo de vida de la aplicación al usar Docker.

Hay muchas razones que aumentarán el número de aplicaciones en contenedores en los próximos años; una de ellas es la creación de aplicaciones basadas en microservicios.

Durante los últimos 15 años, el uso de servicios web ha sido la base de miles de aplicaciones y, probablemente, dentro de unos años estemos en la misma situación con las aplicaciones basadas en microservicios que se ejecutan en contenedores de Docker.

Además, merece la pena mencionar que también se pueden usar contenedores de Docker para aplicaciones monolíticas y disfrutar de la mayoría de las ventajas de Docker. Los contenedores no se centran únicamente en los microservicios.

El uso de los microservicios y la inclusión en contenedores de Docker plantea nuevos desafíos en el proceso de desarrollo de las organizaciones. Por lo tanto, se necesita una estrategia sólida para mantener en ejecución muchos contenedores y microservicios en sistemas de producción. A la larga, las aplicaciones empresariales tendrán cientos e incluso miles de contenedores o instancias que se ejecutan en producción.

Estos desafíos generan nuevas demandas al usar herramientas de DevOps, por lo que tiene que definir nuevos procesos en las actividades de DevOps y encontrar respuestas para el siguiente tipo de preguntas:

- ¿Qué herramientas puedo usar para el desarrollo, CI/CD, la administración y las operaciones?
- ¿Cómo puede mi empresa administrar los errores de los contenedores cuando se ejecutan en producción?
- ¿Cómo se pueden cambiar elementos de software en producción con un tiempo mínimo de inactividad?
- ¿Cómo se puede escalar y supervisar el sistema de producción?
- ¿Cómo se pueden incluir las pruebas y la implementación de contenedores en la canalización de versión?
- ¿Cómo se pueden usar las herramientas y plataformas de código abierto para contenedores en Microsoft Azure?

Si puede responder a todas esas preguntas, estará mejor preparado para mover las aplicaciones (nuevas o

existentes) a contenedores de Docker.

Introducción a un flujo de trabajo genérico de ciclo de vida de aplicaciones de Docker de un extremo a otro

En la figura 2-2 se muestra un flujo de trabajo más detallado del ciclo de vida de aplicaciones de Docker, centrado en este caso en actividades y activos de DevOps específicos.

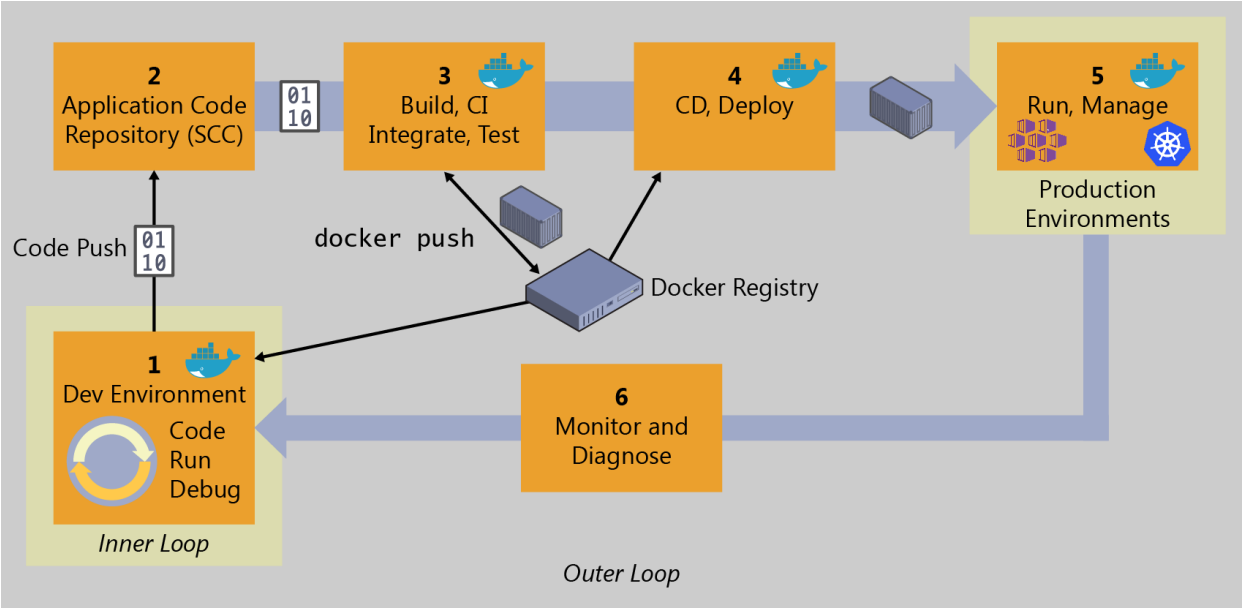


Figura 2-2. Flujo de trabajo de alto nivel para el ciclo de vida de aplicaciones en contenedor de Docker

Todo comienza con el desarrollador, que empieza a escribir código en el flujo de trabajo del bucle interno. La fase de bucle interno es donde los desarrolladores definen todo lo que ocurre antes de insertar código en el repositorio de código (por ejemplo, un sistema de control de código fuente, como GIT). Una vez confirmado, el repositorio desencadena la integración continua (CI) y el resto del flujo de trabajo.

El bucle interno consta de los pasos típicos, como "programación", "ejecución", "prueba" y "depuración", además de los pasos adicionales necesarios justo antes de ejecutar la aplicación en local. Este es el proceso del desarrollador para ejecutar y probar la aplicación como contenedor de Docker. El flujo de trabajo del bucle interno se explicará en las secciones siguientes.

Si se da un paso atrás para observar el flujo de trabajo de un extremo a otro, el flujo de trabajo de DevOps es algo más que una tecnología o un conjunto de herramientas: es una actitud que exige una evolución cultural. Lo integran personas, procesos y herramientas adecuadas que hacen que el ciclo de vida de la aplicación sea más rápido y predecible. Las empresas que adoptan un flujo de trabajo en contenedores suelen reestructurar su organización para que represente a personas y procesos que coincidan con el flujo de trabajo en contenedores.

La práctica con DevOps puede ayudar a los equipos a responder juntos con mayor rapidez ante presiones competitivas, al reemplazar por la automatización los procesos manuales propensos a errores, lo que conlleva una rastreabilidad mejorada y flujos de trabajo repetibles. Las organizaciones también pueden administrar los entornos de manera más eficaz y ahorrar costes gracias a una combinación de recursos locales y en la nube, así como herramientas estrechamente integradas.

Al implementar el flujo de trabajo de DevOps para aplicaciones de Docker, verá que las tecnologías de Docker están presentes en prácticamente todas las etapas del flujo de trabajo, desde el cuadro de desarrollo, durante el trabajo en el bucle interno (programación, ejecución y depuración), en la fase de compilación, prueba y CI y, por último, en la implementación de los contenedores en los entornos de ensayo y producción.

La mejora de las prácticas de calidad ayuda a identificar los defectos al principio del ciclo de desarrollo, lo que reduce el coste que supone corregirlos. Al incluir el entorno y las dependencias en la imagen y adoptar la

filosofía de implementar la misma imagen en varios entornos, se fomenta la disciplina de extraer las configuraciones específicas del entorno, lo cual conlleva implementaciones más fiables.

Los datos enriquecidos que se obtienen a través de la instrumentación efectiva (supervisión y diagnóstico) permiten comprender mejor los problemas de rendimiento y el comportamiento del usuario, algo que resulta muy útil como orientación para las prioridades e inversiones futuras.

DevOps debe considerarse un viaje, no un destino. Debe implementarse de forma incremental a través de proyectos con un ámbito adecuado con los que se pueda demostrar su éxito, aprender y evolucionar.

Ventajas de DevOps para las aplicaciones en contenedor

Estas son algunas de las ventajas más importantes que proporciona un flujo de trabajo sólido de DevOps:

- Entregar software de mayor calidad con más rapidez y un mejor cumplimiento.
- Impulsar la mejora y los ajustes continuos en fases más tempranas y de forma más económica.
- Aumentar la transparencia y la colaboración entre las partes interesadas que participan en la entrega y el funcionamiento del software.
- Controlar los costes y usar recursos aprovisionados de forma más eficaz mientras se minimizan los riesgos de seguridad.
- Conectarse y funcionar automáticamente con muchas de las inversiones existentes de DevOps, incluidas las inversiones en código abierto.

Introducción a la plataforma y las herramientas de Microsoft para aplicaciones en contenedor

02/11/2020 • 10 minutes to read • Edit Online

Visión: Creación de un ciclo de vida de aplicación en contenedor adaptable y de nivel empresarial que abarque el desarrollo, las operaciones de TI y la administración de la producción.

En la figura 3-1 se muestran los principales pilares del ciclo de vida de las aplicaciones de Docker clasificadas por el tipo de trabajo entregado por varios equipos (desarrollo de aplicaciones, procesos de infraestructuras de DevOps y operaciones y administración de TI). Por lo general, en la empresa, los perfiles del "rol" responsable de cada área son diferentes. Sus aptitudes también lo son.

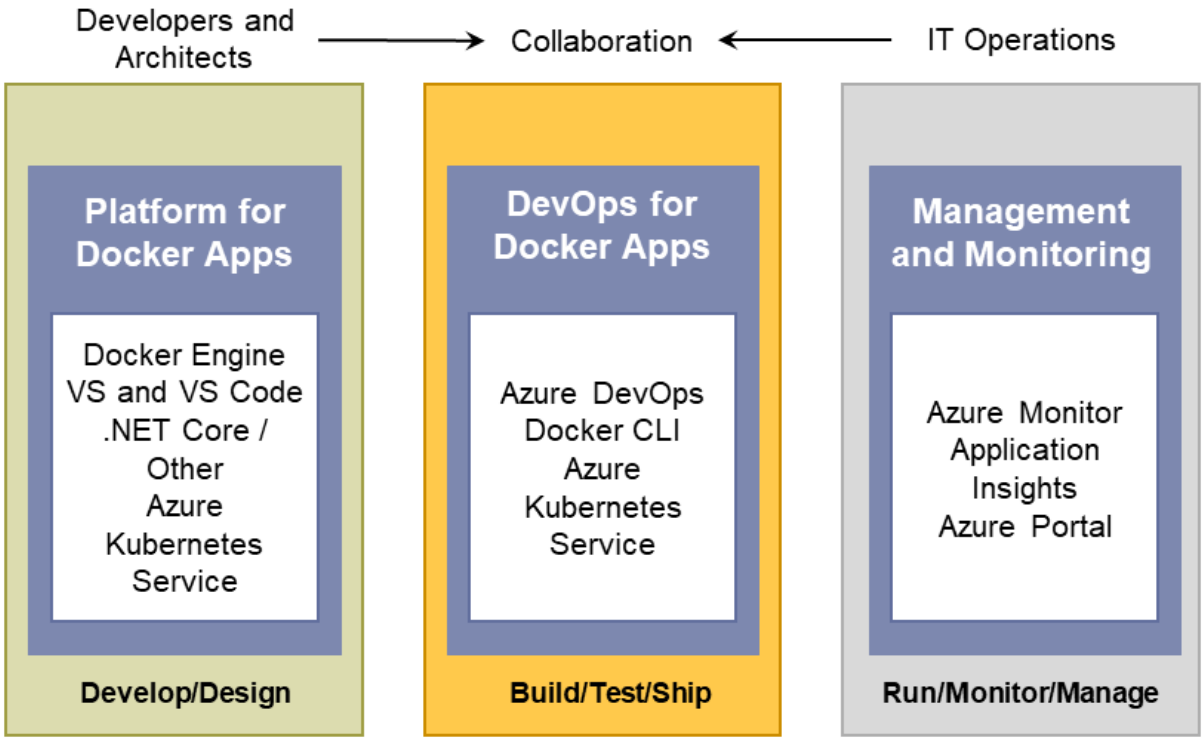


Figura 3-1. Pilares principales del ciclo de vida de las aplicaciones de Docker en contenedor con la plataforma y las herramientas de Microsoft.

Un flujo de trabajo del ciclo de vida de aplicaciones de Docker en contenedor, en principio, puede ser prescriptivo en función de "opciones de productos predeterminadas", de tal forma que los desarrolladores pueden comenzar más rápido, pero es fundamental que, como opción alternativa, haya un marco abierto para que sea un flujo de trabajo flexible capaz de adaptarse a los diferentes contextos de cada organización o empresa. La infraestructura de flujo de trabajo (componentes y productos) debe ser lo suficientemente flexible para abarcar el entorno que cada empresa tendrá en el futuro, con la capacidad incluso de cambiar los productos de desarrollo o DevOps por otros. Esta flexibilidad, receptividad y amplia selección de tecnologías en la plataforma e infraestructura son precisamente las prioridades de Microsoft para aplicaciones de Docker en contenedor, como se explica en los capítulos siguientes.

En la tabla 3-1 se muestra que la intención de las operaciones de Azure DevOps para aplicaciones de Docker en contenedor es ofrecer un flujo de trabajo abierto de DevOps para poder elegir qué productos usar para cada fase (Microsoft o terceros), al mismo tiempo que se ofrece un flujo de trabajo simplificado que proporciona "productos predeterminados" ya conectados; por tanto, puede comenzar rápidamente con el flujo de trabajo de

DevOps de nivel empresarial para aplicaciones de Docker.

Tabla 3-1. Flujos de trabajo de Azure DevOps, abiertos para cualquier tecnología

ADMINISTRADOR DE FLUJOS DE TRABAJO	TECNOLOGÍAS DE MICROSOFT	APLICACIONES DE TERCEROS (ACOPLABLES A AZURE)
Plataforma para aplicaciones de Docker	• Microsoft Visual Studio y Visual Studio Code • .NET • Microsoft Azure Kubernetes Service (AKS) • Azure Container Registry	• Cualquier editor de código (por ejemplo, Sublime) • Cualquier lenguaje (Node.js, Java, Go, etc.) • Cualquier orquestrator y programador • Cualquier registro de Docker
DevOps para aplicaciones de Docker	• Azure DevOps Services • Microsoft Team Foundation Server • Azure Kubernetes Service (AKS)	• GitHub, Git, Subversion, etc. • Jenkins, Chef, Puppet, Velocity, CircleCI, TravisCI, etc. • Centro de datos de Docker local, Kubernetes, Mesos DC/OS, etc.
Administración y supervisión	• Azure Monitor	• Marathon, Chronos, etc.

La plataforma y las herramientas de Microsoft para aplicaciones de Docker en contenedor, como se define en la tabla 3-1, constan de los siguientes componentes:

- **Plataforma de desarrollo de aplicaciones de Docker**. El desarrollo de un servicio una colección de servicios que componen una "aplicación". La plataforma de desarrollo ofrece todo el trabajo que los desarrolladores necesitan antes de insertar el código en un repositorio de código compartido. El desarrollo de servicios, implementados como contenedores, es similar al desarrollo de las mismas aplicaciones o servicios sin Docker. Puede seguir usando su lenguaje favorito (.NET, Node.js, Go, etc.) y su editor o IDE preferidos, como Visual Studio o Visual Studio Code. Sin embargo, en lugar de considerar Docker como un destino de implementación, los servicios se desarrollan en el entorno de Docker. El código se compila, ejecuta, prueba y depura en contenedores a nivel local, proporcionando el entorno de destino en la fase de desarrollo. Al proporcionar el entorno de destino local, los contenedores de Docker configuran lo que ayudará significativamente a mejorar el ciclo de vida de DevOps. Visual Studio y Visual Studio Code disponen de extensiones para integrar los contenedores de Docker en el proceso de desarrollo.

- **DevOps para las aplicaciones de Docker**. Los desarrolladores que crean aplicaciones de Docker pueden usar Azure DevOps o cualquier otro producto de terceros, como Jenkins, para crear una administración integral y automatizada del ciclo de vida de las aplicaciones.

  Con Azure DevOps, los desarrolladores pueden crear DevOps centradas en contenedores para un proceso rápido e iterativo que abarque el control del código fuente desde cualquier plataforma (Azure DevOps-Git, GitHub, cualquier repositorio Git remoto o Subversion), integración continua (CI), pruebas unitarias internas, pruebas de integración entre contenedores y servicios, entrega continua (CD) y administración de versiones. Los desarrolladores también pueden automatizar las versiones de las aplicaciones de Docker en Azure Kubernetes Service (AKS), desde entornos de desarrollo a almacenamiento provisional y producción.

- **Administración y supervisión**. Los equipos de TI pueden administrar y supervisar aplicaciones y servicios de producción de varias maneras, mediante la integración de ambas perspectivas en una experiencia consolidada.

  - **Azure Portal** Azure Kubernetes Service (AKS) ayuda a configurar y mantener los entornos de Docker. También puede usar otros orquestadores para visualizar y configurar el clúster.

- **Herramientas de Docker** Puede administrar las aplicaciones de contenedor con herramientas conocidas. No es necesario cambiar sus prácticas de administración de Docker existentes para mover cargas de trabajo de contenedor a la nube. Use las herramientas de administración de aplicaciones con las que ya está familiarizado y conéctelas a través de los puntos de conexión de API para el orquestador de su elección. También puede usar otras herramientas de terceros para administrar las aplicaciones de Docker o las herramientas de Docker para la CLI.

  Incluso si está familiarizado con los comandos de Linux, puede administrar las aplicaciones de contenedor mediante Microsoft Windows y PowerShell con una línea de comandos del subsistema de Linux y los productos (Docker, Kubernetes, etc.) que los clientes ejecutan en esta funcionalidad del subsistema de Linux. Más adelante en este libro obtendrá más información sobre el uso de estas herramientas en el subsistema de Linux con su sistema operativo favorito de Microsoft Windows.

- **Herramientas de código abierto** Como AKS expone los puntos de conexión de API estándar para el motor de orquestación, las herramientas más populares son compatibles con AKS y, en la mayoría de los casos, son fáciles de usar, incluidos los visualizadores, las herramientas de supervisión, las herramientas de línea de comandos e incluso futuras herramientas a medida que estén disponibles.

- **Azure Monitor** Se trata de una solución de Azure para supervisar todos los aspectos del entorno de producción. Puede supervisar las aplicaciones de Docker en producción con solo configurar su SDK en los servicios, a fin de obtener datos de registro generados por el sistema a partir de las aplicaciones.

Por tanto, Microsoft ofrece una base completa para un ciclo de vida de aplicaciones de Docker en contenedor de un extremo a otro. Se trata de *una colección de productos y tecnologías que le permiten seleccionar, como opción, las herramientas y procesos existentes, así como realizar la integración con ellos*. La flexibilidad de un amplio enfoque y la eficacia de las funcionalidades posicionan a Microsoft como una solución eficaz para el desarrollo de aplicaciones de Docker en contenedor.

Diseño y desarrollo de aplicaciones en contenedor con Docker y Microsoft Azure

02/11/2020 • 2 minutes to read • Edit Online

Visión: Diseño y desarrollo de soluciones escalables con Docker en mente.

Hay muchos casos de uso perfectamente adaptados para los contenedores, no solo para las arquitecturas orientadas a los microservicios, sino también cuando simplemente se dispone de servicios o aplicaciones web comunes que se desean ejecutar y se pretende reducir la fricción entre las implementaciones en entornos de desarrollo y producción.

ANTERIOR SIGUIENTE

Diseño de aplicaciones de Docker

02/11/2020 • 2 minutes to read • Edit Online

En el capítulo 1 se presentaron los conceptos fundamentales relativos a los contenedores y Docker. Se trata del nivel básico de información que necesita para empezar. Con todo, las aplicaciones empresariales pueden ser complejas y componerse de varios servicios en lugar de un solo servicio o contenedor. Para esos casos de uso opcionales, tiene que conocer otros enfoques de diseño, como la arquitectura orientada a servicios (SOA) y los conceptos más avanzados de microservicios y de orquestación de contenedores. El ámbito de este documento no se limita a los microservicios sino que comprende todo el ciclo de vida de aplicaciones de Docker y, por lo tanto, no explora a fondo la arquitectura de microservicios, ya que también se pueden utilizar contenedores y Docker con SOA normales, trabajos o tareas en segundo plano o incluso con enfoques de implementación de aplicaciones monolíticas.

Más información Para obtener más información sobre las aplicaciones empresariales y la arquitectura de microservicios, lea la guía Microservicios de NET: Arquitectura para aplicaciones .NET en contenedor, que también puede descargar en https://aka.ms/MicroservicesEbook.

Pero antes de pasar al ciclo de vida de la aplicación y DevOps, es importante saber cómo se va a diseñar y construir la aplicación y cuáles son las opciones de diseño.

ANTERIOR SIGUIENTE

Principios de diseño de contenedores comunes

02/11/2020 • 2 minutes to read • Edit Online

Antes de entrar en el proceso de desarrollo, hay algunos conceptos básicos que vale la pena mencionar con respecto al uso de los contenedores.

Un contenedor equivale a un proceso

En el modelo de contenedor, un contenedor representa un único proceso. Al definir un contenedor como límite de proceso, se empiezan a crear los tipos primitivos que se utilizan en procesos de escalado o generación de lotes. Al ejecutar un contenedor de Docker, verá una definición ENTRYPOINT. Esta define el proceso y la duración del contenedor. Cuando se completa el proceso, finaliza el ciclo de vida del contenedor. Hay procesos de larga ejecución (como servidores web) y procesos de corta duración (como trabajos por lotes), que posiblemente se hayan implementado como Microsoft Azure WebJobs. Si se produce un error en el proceso, el contenedor finaliza y lo sustituye el orquestador. Si se indicó al orquestador que mantuviera cinco instancias en ejecución y se produce un error en una de ellas, el orquestador creará otro contenedor para reemplazar al proceso erróneo. En un trabajo por lotes, el proceso se inicia con parámetros. Cuando el proceso finalice, el trabajo se habrá completado.

Es posible que en algún momento le interese que varios procesos se ejecuten en un solo contenedor. En cualquier documento de arquitectura, nunca hay un "nunca" ni siempre hay un "siempre". Para los escenarios que requieren varios procesos, un patrón común es usar Supervisor.

Aplicaciones monolíticas

06/03/2021 • 11 minutes to read • Edit Online

En este escenario, compilará una aplicación o un servicio web único y monolítico y lo implementará como un contenedor. La estructura de la aplicación podría no ser monolítica, sino incluir varias bibliotecas, componentes o incluso capas (nivel de aplicación, capa de dominio, capa de acceso a datos, etc.). Externamente es un contenedor único, como un proceso único, una aplicación web única o un servicio único.

Para administrar este modelo, debe implementar un único contenedor para representar la aplicación. Para escalarlo, solo tiene que agregar unas pocas copias más con un equilibrador de carga delante. La simplicidad proviene de administrar una única implementación en un solo contenedor o máquina virtual.

Al seguir el principio de que un contenedor realiza una sola acción y lo hace en un proceso, el patrón monolítico entra en conflicto. Puede incluir varios componentes, bibliotecas o capas internas en cada contenedor, como se muestra en la figura 4-1.

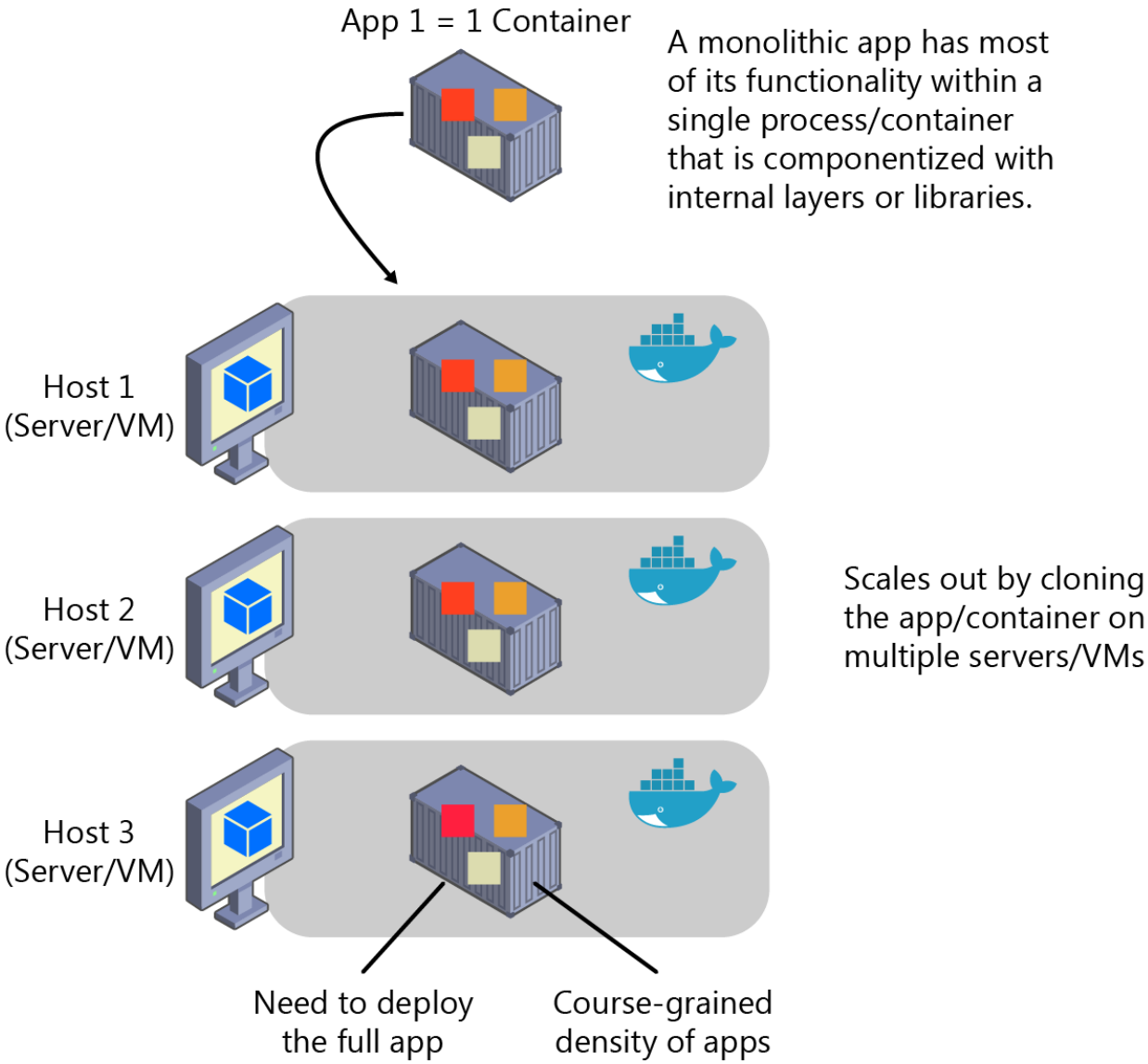


Figura 4-1. Ejemplo de arquitectura de aplicaciones monolíticas

Una aplicación monolítica tiene todas o la mayoría de sus funcionalidades en un único proceso o contenedor y está formada por capas internas o bibliotecas. El inconveniente de este enfoque aparece cuando la aplicación aumenta y debe escalarse. Si se escala toda la aplicación, realmente no es un problema. Aun así, en la mayoría

de los casos, solo unos pocos elementos de la aplicación son los puntos de obstrucción que deben escalarse, mientras que otros componentes se usan menos.

En el ejemplo típico de comercio electrónico, lo que probablemente necesite es escalar el componente de información del producto. Hay muchos más clientes que buscan productos de los que los compran. Más clientes usan la cesta en lugar de usar la canalización de pago. Menos clientes publican comentarios o consultan su historial de compras. Y es probable que solo cuente con un grupo reducido de empleados, en una única región, que tenga que administrar las campañas de contenido y marketing. Al escalar el diseño monolítico, todo el código se implementa varias veces.

Además del problema de "escalarlo todo", los cambios en un único componente requieren volver a probar por completo toda la aplicación e implementar por completo todas las instancias.

El enfoque monolítico es habitual y muchas organizaciones realizan el desarrollo con este método de diseño. Muchas disfrutan de resultados bastante buenos, mientras que otras se enfrentan a algunos límites. Muchas diseñaron sus aplicaciones con este modelo, ya que crear arquitecturas orientadas a servicios (SOA) con infraestructura y herramientas resultaba demasiado difícil, y no vieron la necesidad hasta que la aplicación creció.

Desde la perspectiva de la infraestructura, cada servidor puede ejecutar muchas aplicaciones dentro del mismo host y tener una proporción aceptable de eficacia en el uso de recursos, como se muestra en la figura 4-2.

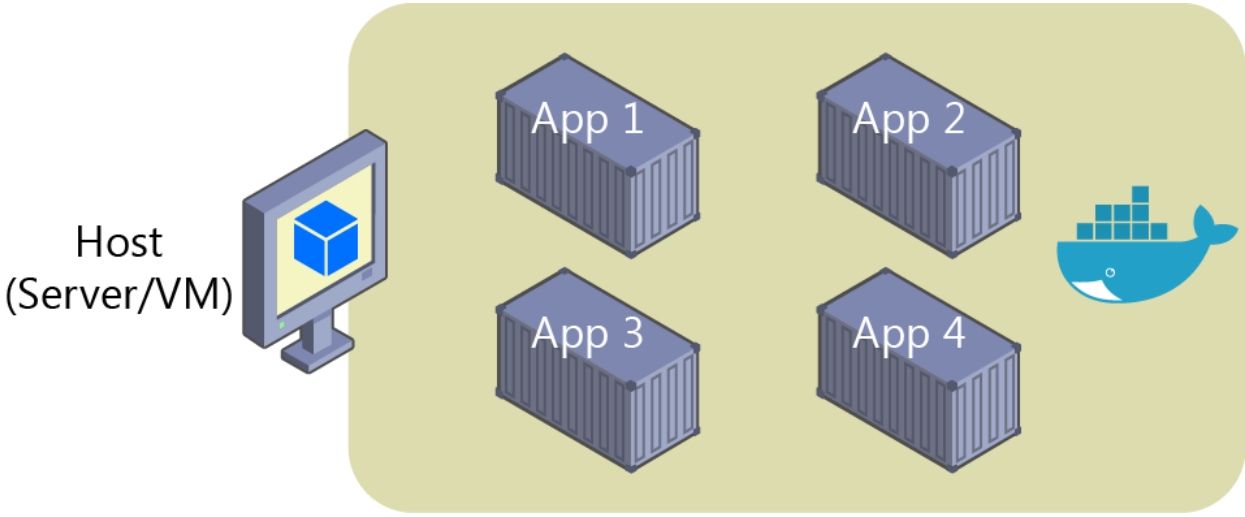


Figura 4-2. Host que ejecuta varias aplicaciones o contenedores

Por último, desde la perspectiva de la disponibilidad, las aplicaciones monolíticas deben implementarse como un conjunto. Esto significa que, en caso de que deba *detener e iniciar*, todas las funcionalidades y todos los usuarios se verán afectados durante el período de implementación. En determinadas situaciones, el uso de Azure y de contenedores puede minimizar estas situaciones y reducir la probabilidad de tiempo de inactividad en la aplicación, como se aprecia en la figura 4-3.

Las aplicaciones monolíticas pueden implementarse en Azure con máquinas virtuales dedicadas para cada instancia. Con Azure VM Scale Sets, las máquinas virtuales se pueden escalar fácilmente.

También puede usar Azure App Services para ejecutar aplicaciones monolíticas y escalar fácilmente instancias sin necesidad de administrar las máquinas virtuales. Azure App Services también puede ejecutar instancias únicas de contenedores de Docker, lo que simplifica la implementación.

Puede implementar varias máquinas virtuales como hosts de Docker y ejecutar cualquier número de contenedores por máquina virtual. Después, puede administrar el escalado mediante Azure Load Balancer, como se muestra en la figura 4-3.

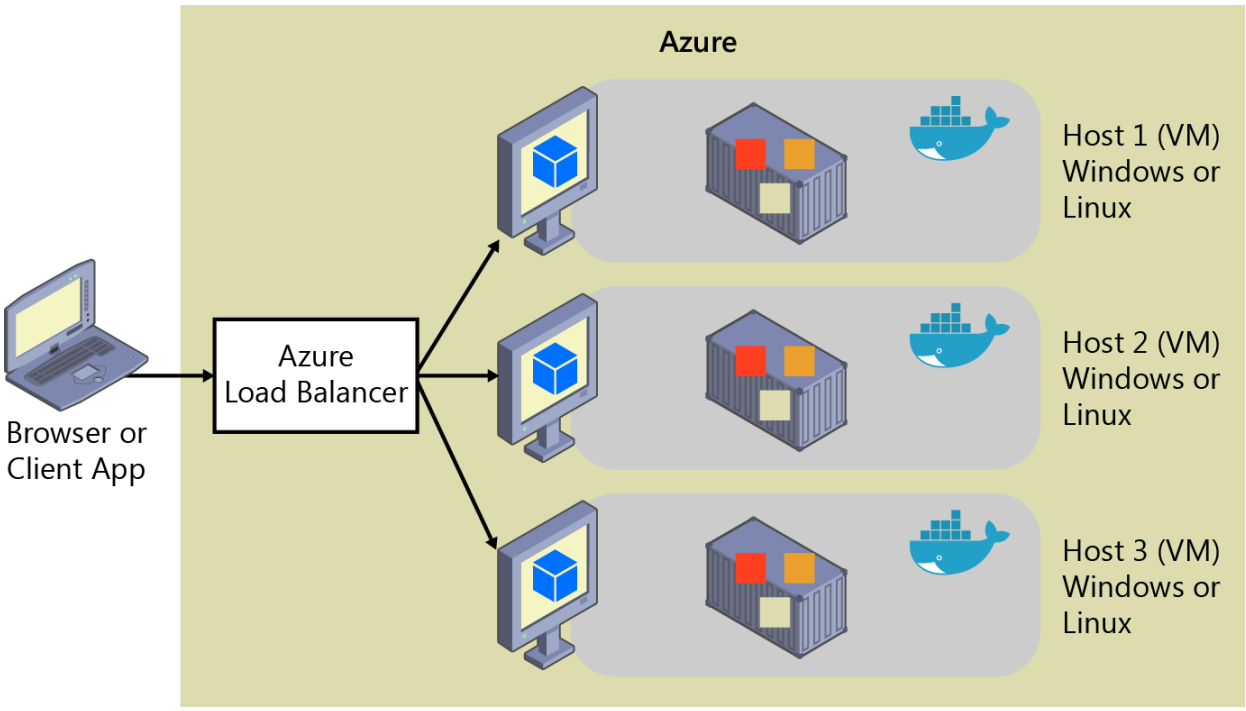


Figura 4-3. Varios hosts escalan horizontalmente una única aplicación de Docker

Puede administrar la implementación de los propios hosts mediante técnicas de implementación tradicionales.

Puede administrar contenedores de Docker desde la línea de comandos mediante el uso de comandos como `docker run` y `docker-compose up`. También puede automatizar el proceso en las canalizaciones de entrega continua (CD) y realizar la implementación en hosts de Docker desde Azure DevOps Services, por ejemplo.

Una aplicación monolítica implementada como un contenedor

El uso de contenedores para administrar implementaciones monolíticas tiene una serie de ventajas. Escalar las instancias de los contenedores es mucho más rápido y fácil que implementar máquinas virtuales adicionales.

Implementar las actualizaciones como imágenes de Docker es mucho más rápido y eficaz en la red. Normalmente, los contenedores de Docker se inician en segundos, lo que acelera las implementaciones. Anular un contenedor de Docker es tan fácil como invocar el comando `docker stop`, que normalmente se completa en menos de un segundo.

Como por diseño los contenedores son intrínsecamente inmutables, no tendrá que preocuparse de que las máquinas virtuales resulten dañadas porque un script de actualización olvidó tener en cuenta alguna configuración concreta o archivo que se conserve en disco.

Si bien las aplicaciones monolíticas pueden beneficiarse de Docker, estamos examinando estos beneficios muy por encima. Las ventajas adicionales de administrar contenedores proceden de implementar con orquestadores de contenedor que administran las distintas instancias y el ciclo de vida de cada instancia del contenedor. Separar la aplicación monolítica en subsistemas que se pueden escalar, desarrollar e implementar de forma individual es el punto de entrada al reino de los microservicios.

Para obtener información sobre cómo migrar aplicaciones monolíticas con contenedores mediante lift-and-shift y cómo puede modernizar las aplicaciones, lea la guía adicional de Microsoft titulada Modernización de las aplicaciones .NET existentes con la nube de Azure y contenedores de Windows, que también puede descargar como PDF desde https://aka.ms/LiftAndShiftWithContainersEbook.

Publicación de una única aplicación de contenedor de Docker en Azure App Service

Tanto si quiere obtener la validación rápida de un contenedor implementado en Azure como si la aplicación es simplemente una aplicación de un solo contenedor, Azure App Service ofrece una excelente manera de proporcionar servicios escalables de un solo contenedor.

El uso de Azure App Service es intuitivo, por lo que puede empezar a trabajar rápidamente. Gracias a su excelente integración con GIT, puede tomar el código, compilarlo en Microsoft Visual Studio e implementarlo directamente en Azure. Tradicionalmente (es decir, sin Docker), si necesitaba otras funcionalidades, marcos o dependencias que no son compatibles con App Service, debía esperar a que el equipo de Azure actualizase esas dependencias en App Service o cambiar a otros servicios como Service Fabric, Cloud Services o incluso simples máquinas virtuales, sobre las que tiene más control y en las que puede instalar un componente o un marco necesarios para la aplicación.

Ahora, como se muestra en la figura 4-4, al usar Visual Studio 2019, la compatibilidad con contenedores en Azure App Service le ofrece la capacidad de incluir todo lo que quiera en el entorno de aplicación. Si agregó una dependencia en la aplicación porque la ejecuta en un contenedor, podrá incluir esas dependencias en la imagen de Docker o en el archivo Dockerfile.

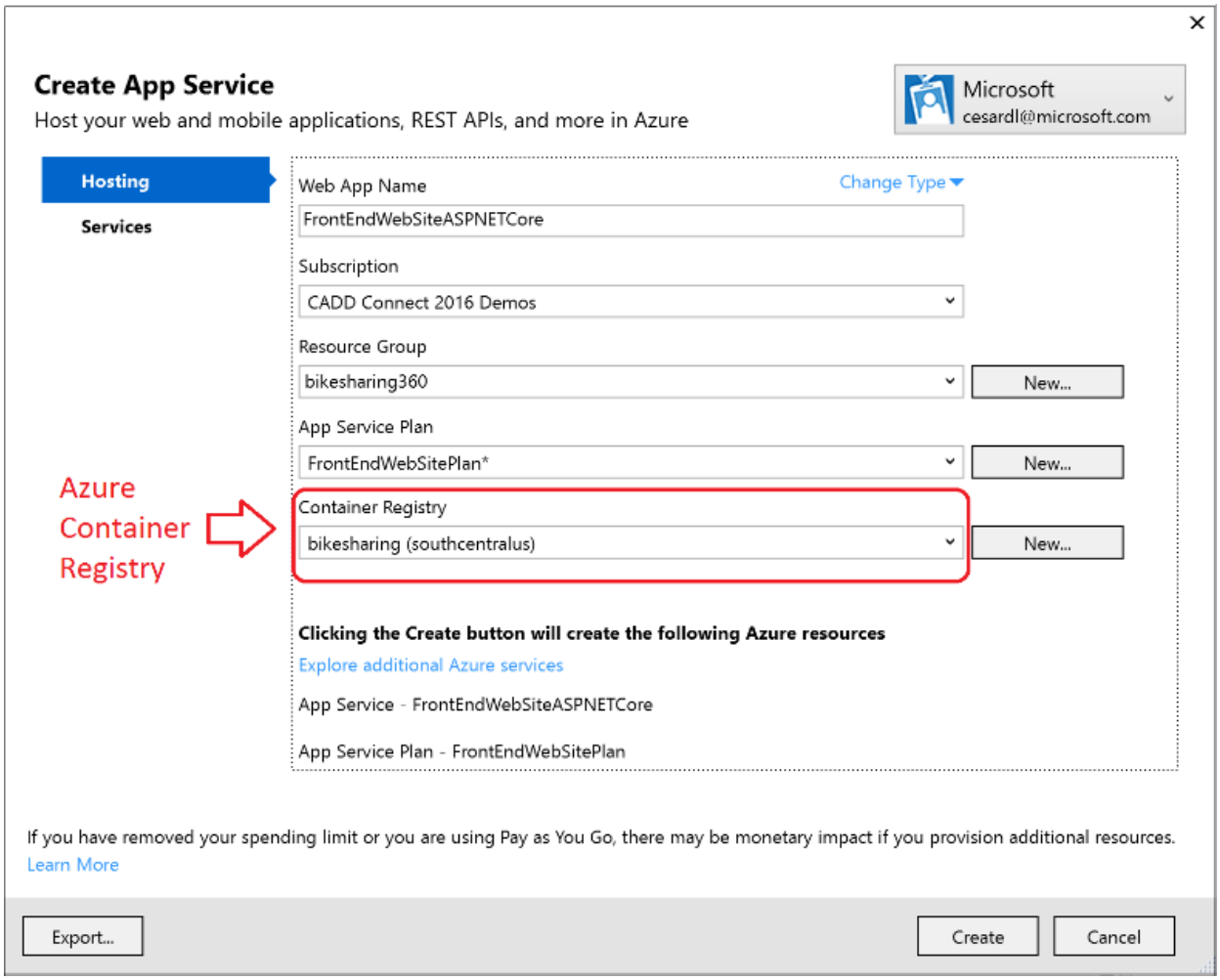


Figura 4-4. Publicación de un contenedor en Azure App Service desde aplicaciones o contenedores de Visual Studio

En la figura 4-4 también puede verse que el flujo de publicación inserta una imagen a través de un registro de contenedor, que puede ser Azure Container Registry (un registro cercano a las implementaciones en Azure y protegido por las cuentas y los grupos de Azure Active Directory) o cualquier otro registro de Docker, como Docker Hub o registros locales.

Estado y datos en aplicaciones de Docker

02/11/2020 • 10 minutes to read • Edit Online

En la mayoría de los casos, un contenedor se puede considerar como una instancia de un proceso. Un proceso no mantiene un estado persistente. Si bien un contenedor puede escribir en su almacenamiento local, suponer que una instancia permanecerá indefinidamente es como suponer que una sola ubicación en memoria será duradera. Debe suponerse que las imágenes del contenedor, como los procesos, tienen varias instancias y que finalmente desaparecerán; si se administran con un orquestador de contenedores, se debe suponer que podrían desplazarse de un nodo o máquina virtual a otro.

Las soluciones siguientes se usan para administrar datos persistentes en aplicaciones de Docker:

Desde el host de Docker, como volúmenes de Docker:

- Los **volúmenes** se almacenan en un área del sistema de archivos de host administrado por Docker.
- Los **montajes de enlace** pueden asignarse a cualquier carpeta del sistema de archivos de host, por lo que el acceso no se puede controlar desde el proceso de Docker y puede suponer un riesgo de seguridad ya que un contenedor podría acceder a carpetas del sistema operativo confidenciales.
- Los **montajes tmpfs** son como carpetas virtuales que solo existen en la memoria del host y nunca se escriben en el sistema de archivos.

Desde el almacenamiento remoto:

- Azure Storage proporciona almacenamiento con distribución geográfica y representa una buena solución de persistencia a largo plazo para los contenedores.
- Bases de datos relacionales remotas como Azure SQL Database, bases de datos NoSQL como Azure Cosmos DB o servicios de caché como Redis.

Desde el contenedor de Docker:

- Docker ofrece una característica denominada el *sistema de archivos de superposición*. Esta característica implementa una tarea de copia en escritura que almacena información actualizada en el sistema de archivos raíz del contenedor. Esa información "se coloca encima" de la imagen original en la que se basa el contenedor. Si se elimina el contenedor del sistema, estos cambios se pierden. Por tanto, si bien es posible guardar el estado de un contenedor dentro de su almacenamiento local, diseñar un sistema sobre esta base entraría en conflicto con la idea del diseño del contenedor, que de manera predeterminada es sin estado.
- Sin lugar a duda, los volúmenes de Docker son ahora la mejor manera de controlar datos locales en Docker. Si necesita obtener más información sobre el almacenamiento en contenedores, consulte Docker storage drivers (Controladores de almacenamiento de Docker) y About images, containers, and storage drivers (Acerca de las imágenes, los contenedores y los controladores de almacenamiento).

En los siguientes puntos se ofrece más información sobre estas opciones.

Los **volúmenes** son directorios asignados desde el sistema operativo del host a directorios en contenedores. Cuando el código en el contenedor tiene acceso al directorio, ese acceso es realmente a un directorio en el sistema operativo del host. Este directorio no está asociado a la duración del contenedor y Docker lo administra y aísla de la funcionalidad básica de la máquina host. Por tanto, los volúmenes de datos están diseñados para conservar los datos independientemente de la vida del contenedor. Si elimina un contenedor o una imagen del host de Docker, los datos que se conservan en el volumen de datos no se eliminan.

Los volúmenes pueden tener nombre o ser anónimos (predeterminado). Los volúmenes con nombre son la evolución de los **Contenedores de volúmenes de datos** y facilitan el uso compartido de datos entre contenedores. Los volúmenes también admiten controladores de volumen, que le permiten almacenar datos en hosts remotos, entre otras opciones.

Los **montajes de enlace** han estado disponibles durante mucho tiempo y permiten la asignación de cualquier carpeta a un punto de montaje en un contenedor. Los montajes de enlace tienen más limitaciones que los volúmenes y algunos problemas de seguridad importantes, por lo que los volúmenes son la opción recomendada.

Los **montajes** `tmpfs` son carpetas virtuales que solo existen en la memoria del host y nunca se escriben en el sistema de archivos. Son rápidos y seguros, pero usan memoria y solo están diseñados para datos no persistentes.

Tal como se muestra en la figura 4-5, los volúmenes de Docker normales pueden almacenarse fuera de los propios contenedores, pero dentro de los límites físicos del servidor de host o de la máquina virtual. Sin embargo, los contenedores de Docker no pueden tener acceso a un volumen desde un servidor de host o máquina virtual a otro. En otras palabras, con estos volúmenes, no es posible administrar los datos que se comparten entre contenedores que se ejecutan en otros hosts de Docker, aunque se podría lograr con un controlador de volumen que sea compatible con los hosts remotos.

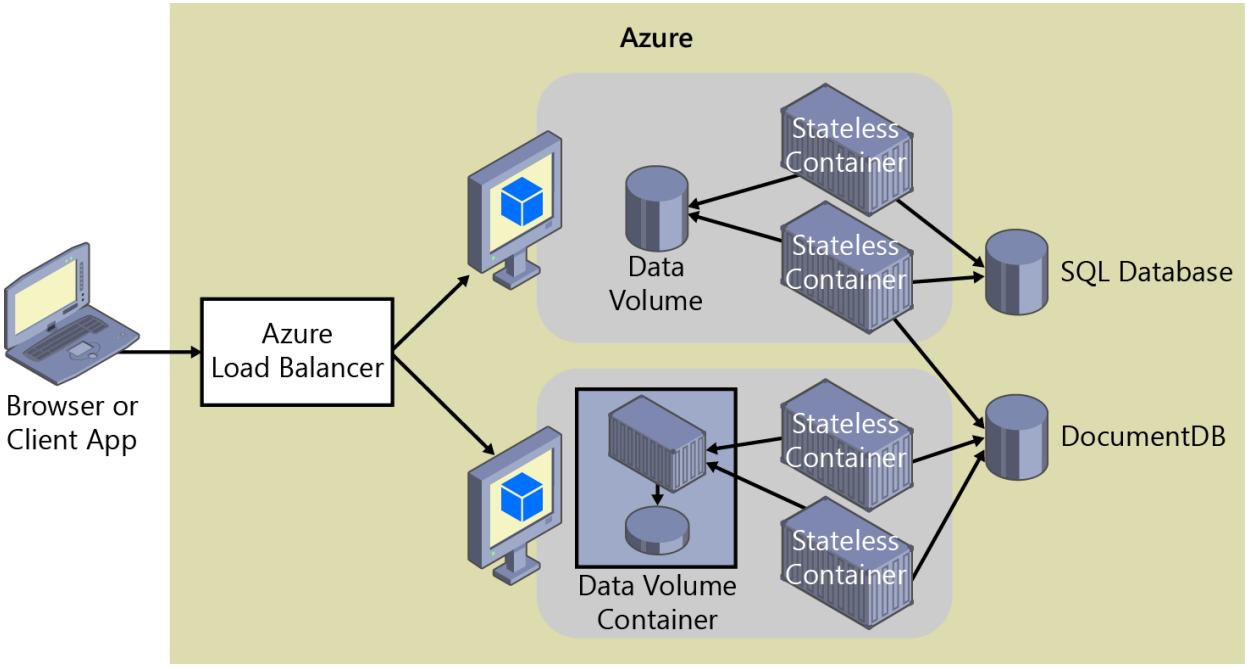


Figura 4-5. Volúmenes y orígenes de datos externos para aplicaciones basadas en contenedor

Además, cuando un orquestador administra los contenedores de Docker, estos podrían "moverse" entre hosts, según las optimizaciones que el clúster realice. Por tanto, no se recomienda usar volúmenes de datos para los datos empresariales. Pero son un buen mecanismo para trabajar con archivos de seguimiento, archivos temporales o similares que no afectarán a la coherencia de los datos empresariales.

Las herramientas de **orígenes de datos remotos y caché** como Azure SQL Database, Azure Cosmos DB o una caché remota como Redis pueden usarse en aplicaciones en contenedores del mismo modo que se usan al desarrollar sin contenedores. Se trata de una manera comprobada para almacenar datos de aplicaciones empresariales.

Azure Storage. Normalmente, los datos empresariales deben colocarse en recursos o bases de datos externos, como Azure Storage. Azure Storage ofrece los siguientes servicios en la nube:

- El almacenamiento de blobs almacena datos de objetos no estructurados. Un blob puede ser cualquier tipo de texto o datos binarios, como documentos o archivos multimedia (archivos de imagen, audio y

vídeo). El almacenamiento de blobs también se conoce como "almacenamiento de objetos".

- File Storage ofrece almacenamiento compartido para aplicaciones heredadas que usan el protocolo SMB estándar. Las máquinas virtuales de Azure y los servicios en la nube pueden compartir datos de archivos en los componentes de la aplicación a través de recursos compartidos montados. Las aplicaciones locales pueden acceder a datos de archivos en un recurso compartido a través de la API REST de File Service.
- El almacenamiento de tabla almacena conjuntos de datos estructurados. El almacenamiento de tabla es un almacén de datos del atributo de clave NoSQL, lo que permite desarrollar y acceder rápidamente a grandes cantidades de datos.

Bases de datos relacionales y bases de datos NoSQL. Existen muchas opciones de bases de datos externas, desde las bases de datos relacionales como SQL Server, PostgreSQL u Oracle, o las bases de datos NoSQL como Azure Cosmos DB, MongoDB, etc. En esta guía no se explicarán estas bases de datos ya que se trata de un tema totalmente diferente.

Aplicaciones orientadas a servicios

02/11/2020 • 2 minutes to read • Edit Online

Arquitectura orientada a servicios (SOA) era un término muy utilizado que significaba muchas cosas diferentes para diferentes personas. Pero, como denominador común, SOA significa que la arquitectura de una aplicación se estructura descomponiéndola en varios servicios (normalmente como servicios HTTP) que se pueden clasificar en tipos diferentes como subsistemas o, en otros casos, como niveles.

Esos servicios se pueden implementar ahora como contenedores de Docker, con lo que se resuelven los problemas de implementación, puesto que todas las dependencias se incluyen en la imagen de contenedor. De todos modos, cuando necesite escalar arquitecturas orientadas a servicios, puede que se encuentre con problemas si realiza una implementación en función de instancias únicas. Este problema puede controlarse con software de agrupación en clústeres de Docker o un orquestador. Los orquestadores se van a examinar más detalladamente en la sección siguiente, cuando se analicen los enfoques de microservicios.

Los contenedores de Docker son útiles (pero no obligatorios) para las arquitecturas orientadas a servicios tradicionales y las arquitecturas de microservicios más avanzadas.

En definitiva, las soluciones de agrupación en clústeres de contenedores resultan útiles tanto para una arquitectura tradicional de SOA como para una arquitectura de microservicios más avanzada en la que cada microservicio posee su modelo de datos. Y, gracias a varias bases de datos, también puede escalar horizontalmente el nivel de datos en lugar de trabajar con bases de datos monolíticas compartidas por los servicios SOA. No obstante, la discusión sobre la división de los datos se centra exclusivamente en la arquitectura y el diseño.

ANTERIOR SIGUIENTE

Orquestación de microservicios y aplicaciones de varios contenedores para una alta escalabilidad y disponibilidad

02/11/2020 • 35 minutes to read • Edit Online

La utilización de orquestadores para aplicaciones listas para producción es fundamental si la aplicación se basa en microservicios o está dividida entre varios contenedores. Como se mencionó anteriormente, en un enfoque basado en microservicios, cada microservicio posee su modelo y sus datos para que sea autónomo desde un punto de vista del desarrollo y la implementación. Pero incluso si tiene una aplicación más tradicional que se compone de varios servicios (por ejemplo, SOA), también tendrá varios contenedores o servicios que conforman una sola aplicación de negocio que deban implementarse como un sistema distribuido. Estos tipos de sistemas son difíciles de administrar y escalar horizontalmente; por lo tanto, un orquestador es indispensable si se quiere tener una aplicación de varios contenedores, escalable y lista para la producción.

La figura 4-6 ilustra la implementación en un clúster de una aplicación formada por varios microservicios (contenedores).

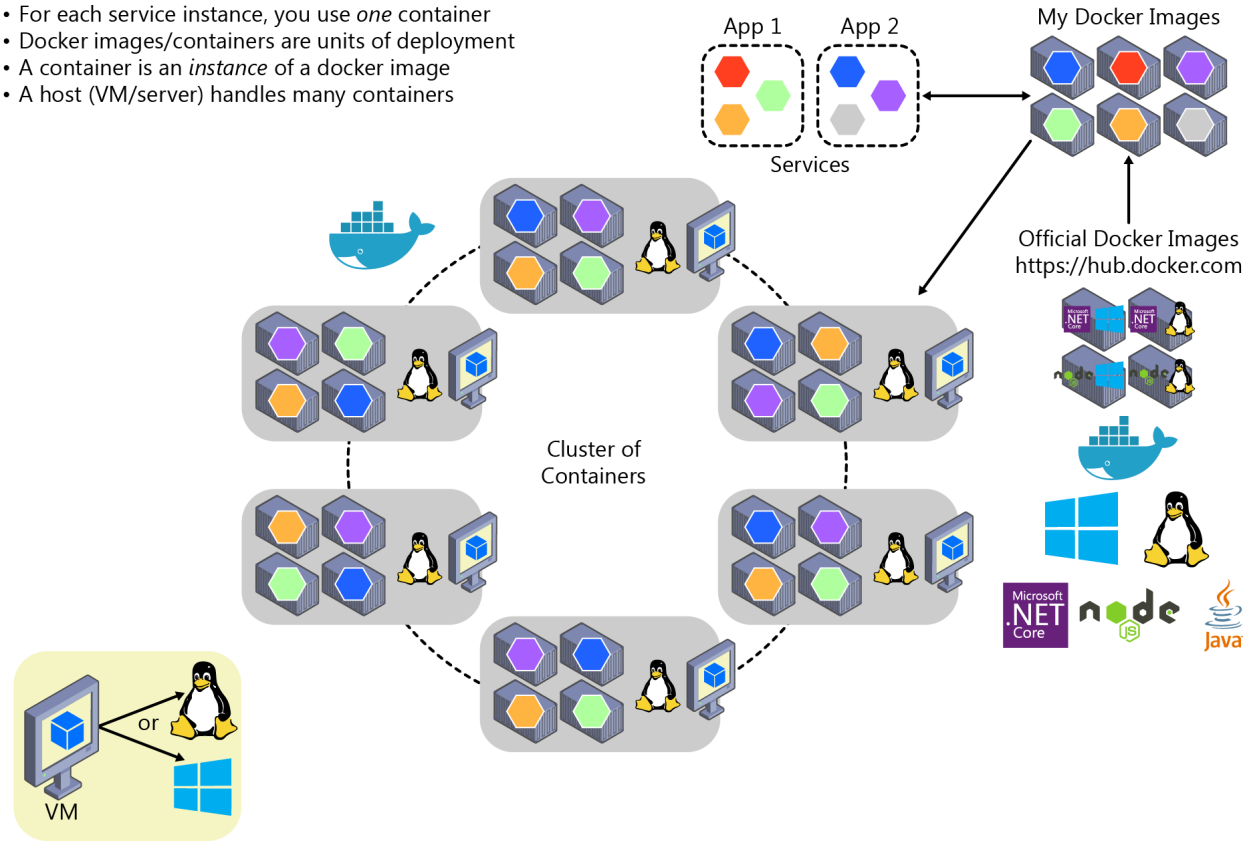


Figura 4-6. Un clúster de contenedores

Parece un enfoque lógico. Pero ¿cómo se está administrando el equilibrio de carga, el enrutamiento y la orquestación de estas aplicaciones compuestas?

La CLI de Docker satisface las necesidades de administración de un contenedor en un host, pero se queda corta a la hora de administrar varios contenedores implementados en varios hosts para aplicaciones distribuidas más complejas. En la mayoría de los casos, se necesita una plataforma de administración que automáticamente inicie los contenedores, escale horizontalmente los contenedores con varias instancias por imagen, los suspenda o los cierre cuando sea necesario y, a ser posible, también controle su acceso a recursos como la red y el

almacenamiento de datos.

Para ir más allá de la administración de contenedores individuales o aplicaciones compuestas simples y pasar a aplicaciones empresariales más grandes con microservicios, debe cambiar a orquestación y plataformas de agrupación en clústeres.

Desde el punto de vista de la arquitectura y el desarrollo, si está compilando grandes aplicaciones empresariales basadas en microservicios, es importante familiarizarse con las siguientes plataformas y productos que admiten escenarios avanzados:

- **Clústeres y orquestadores**. Cuando se necesita escalar horizontalmente las aplicaciones a varios hosts de Docker, como con una aplicación grande basada en microservicios, es fundamental poder administrar todos los hosts como un solo clúster mediante la abstracción de la complejidad de la plataforma subyacente. Eso es lo que proporcionan los clústeres de contenedor y los orquestadores. Azure Service Fabric y Kubernetes son ejemplos de orquestadores. Kubernetes está disponible en Azure a través de Azure Kubernetes Service.
- **Programadores**. *Programar* significa tener la capacidad de que un administrador inicie los contenedores de un clúster, por lo que los programadores también proporcionan una interfaz de usuario para hacerlo. Un programador de clúster tiene varias responsabilidades: usar eficazmente los recursos del clúster, establecer las restricciones definidas por el usuario, equilibrar eficazmente la carga de los contenedores entre los distintos nodos o hosts, ser resistente a los errores y proporcionar un alto grado de disponibilidad.

Los conceptos de un clúster y un programador están estrechamente relacionados, por lo que los productos proporcionados por diferentes proveedores suelen ofrecer ambos conjuntos de funciones. En la sección siguiente se muestran las plataformas y las opciones de software más importantes disponibles para clústeres y programadores. Estos orquestadores generalmente se ofrecen en nubes públicas como Azure.

Plataformas de software para agrupación en clústeres de contenedores, orquestación y programación

PLATAFORMA	COMENTARIOS
Kubernetes 	*Kubernetes* es un producto de código abierto cuya funcionalidad abarca desde la infraestructura de clúster y la programación de contenedores a las capacidades de orquestación. Permite automatizar la implementación, la escala y las operaciones de contenedores de aplicaciones en varios clústeres de hosts. *Kubernetes* proporciona una infraestructura centrada en el contenedor que agrupa los contenedores de la aplicación en unidades lógicas para facilitar la administración y detección. *Kubernetes* está más desarrollado en Linux que en Windows.
Azure Kubernetes Service (AKS) 	Azure Kubernetes Service (AKS) es un servicio de orquestación de contenedores de Kubernetes administrado en Azure que simplifica la administración, implementación y operaciones del clúster de Kubernetes.

PLATAFORMA	COMENTARIOS
Azure Service Fabric 	Service Fabric es una plataforma de microservicios de Microsoft para crear aplicaciones. Es un orquestador de servicios y crea clústeres de máquinas. Service Fabric puede implementar servicios como contenedores o procesos estándar. Puede incluso combinar servicios en procesos con servicios en contenedores dentro de la misma aplicación y el mismo clúster. Los clústeres de *Service Fabric* pueden implementarse en Azure, de forma local o en cualquier nube. Con todo, la implementación se simplifica en Azure con un enfoque administrado. *Service Fabric* proporciona modelos de programación de Service Fabric prescriptivos adicionales y opcionales como servicios con estado y Reliable Actors. *Service Fabric* está más desarrollado en Windows (con años de evolución en Windows) que en Linux. Tanto los contenedores de Linux como los de Windows son compatibles con Service Fabric desde 2017.
Azure Service Fabric Mesh 	*Azure Service Fabric Mesh* ofrece el mismo nivel de fiabilidad, rendimiento crítico y escala que Service Fabric, pero también proporciona una plataforma totalmente administrada y sin servidor. No es necesario administrar un clúster, las máquinas virtuales, el almacenamiento o la configuración de red. Puede centrarse en el desarrollo de la aplicación. *Service Fabric Mesh* admite contenedores de Windows y Linux, lo que permite desarrollar con cualquier lenguaje de programación y marco de trabajo que elija.

Uso de orquestadores basados en contenedor en Azure

Existen varios proveedores de nube que ofrecen compatibilidad con contenedores de Docker más compatibilidad con la orquestación y los clústeres de Docker, como Azure, Amazon EC2 Container Service y Google Container Engine. Azure proporciona compatibilidad con el orquestador y el clúster de Docker a través de Azure Kubernetes Service (AKS), Azure Service Fabric y Azure Service Fabric Mesh.

Uso de Azure Kubernetes Service

Un clúster de Kubernetes agrupa varios hosts de Docker y los expone como un único host virtual de Docker, lo que permite implementar varios contenedores en el clúster y escalar horizontalmente con cualquier número de instancias de contenedor. El clúster controlará toda la mecánica de administración compleja, como la escalabilidad, el estado, etc.

AKS proporciona una manera de simplificar la creación, la configuración y la administración de un clúster de máquinas virtuales en Azure que están preconfiguradas para ejecutar aplicaciones en contenedores. Al utilizar una configuración optimizada de herramientas de orquestación y programación de código abierto populares, AKS le permite usar sus habilidades existentes o aprovechar un gran corpus creciente de conocimientos de la comunidad para implementar y administrar aplicaciones basadas en contenedor en Microsoft Azure.

Azure Kubernetes Service optimiza la configuración de tecnologías y herramientas populares de código abierto de agrupación en clústeres de Docker específicamente para Azure. Se trata de una solución abierta que ofrece la portabilidad de los contenedores y la configuración de la aplicación. Seleccione el tamaño, el número de hosts y

las herramientas de orquestador, y AKS se encarga de todo lo demás.

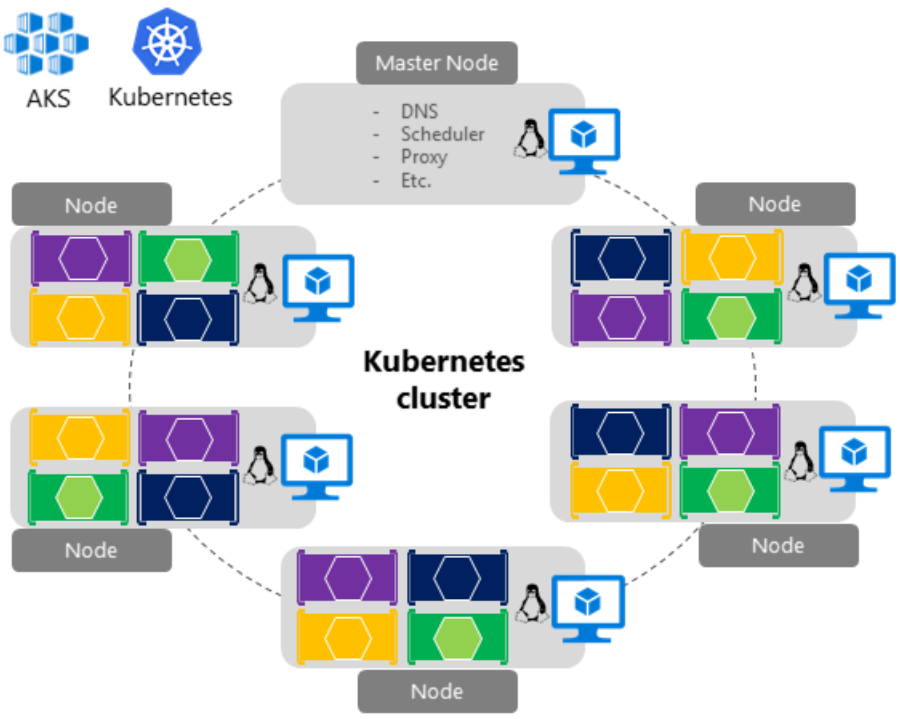


Figura 4-7. Topología y estructura simplificada del clúster de Kubernetes

En la figura 4-7 se muestra la estructura de un clúster de Kubernetes, donde un nodo maestro (VM) controla la mayor parte de la coordinación del clúster y puede implementar contenedores en el resto de los nodos que se administran como un único grupo desde el punto de vista de la aplicación. Esto le permite escalar a miles o incluso a decenas de miles de contenedores.

Entorno de desarrollo para Kubernetes

En el entorno de desarrollo, Docker anunció en julio de 2018 que Kubernetes también puede ejecutarse en un único equipo de desarrollo (Windows 10 o macOS). Basta con instalar Docker Desktop. Puede implementar posteriormente en la nube (AKS) para obtener más pruebas de integración, como se muestra en la figura 4-8.

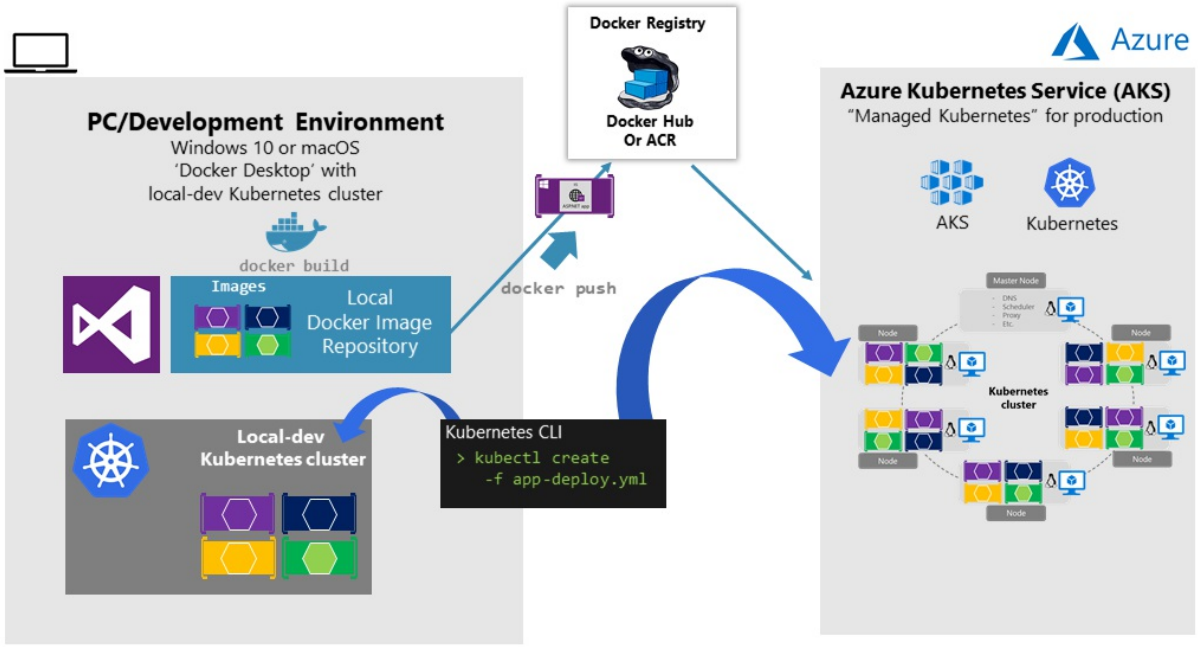


Figura 4-8. Ejecución de Kubernetes en el equipo de desarrollo y la nube

Introducción a Azure Kubernetes Service (AKS)

Para empezar a usar AKS, implemente un clúster de AKS desde Azure Portal o mediante la CLI. Para obtener más información sobre la implementación de un clúster de Kubernetes en Azure, vea Inicio rápido: Implementación de un clúster de Azure Kubernetes Service (AKS).

No hay cuotas para el software instalado de forma predeterminada como parte de AKS. Todas las opciones predeterminadas se implementan con el software de código abierto. AKS está disponible en varias máquinas virtuales en Azure. Se cobra únicamente por las instancias de proceso que se elijan, así como por los otros recursos subyacentes de la infraestructura que se utilicen como, por ejemplo, la red y el almacenamiento. No hay ningún cargo incremental para AKS.

Para obtener más información sobre la implementación en Kubernetes en función de `kubectl` y archivos `.yaml` originales, vea Implementación en Azure Kubernetes Service (AKS).

Implementación con gráficos de Helm en clústeres de Kubernetes

Al implementar una aplicación en un clúster de Kubernetes, puede usar la herramienta de CLI `kubectl.exe` original mediante archivos de implementación basados en el formato nativo (archivos `.yaml`), como ya se mencionó en la sección anterior. Pero, para aplicaciones de Kubernetes más complejas, como al implementar aplicaciones complejas basadas en microservicios, se recomienda usar Helm.

Los gráficos de Helm le ayudan a definir, establecer la versión, instalar, compartir, actualizar o revertir incluso la aplicación más compleja de Kubernetes.

Adicionalmente, el uso de Helm se recomienda porque otros entornos de Kubernetes en Azure, como Azure Dev Spaces, también se basan en los gráficos de Helm.

La Cloud Native Computing Foundation (CNCF) mantiene Helm en colaboración con Microsoft, Google, Bitnami y la comunidad de colaboradores de Helm.

Para obtener más información sobre la implementación en gráficos de Helm y Kubernetes, vea la sección Instalación de eShopOnContainers mediante Helm.

Uso de Azure Dev Spaces para el ciclo de vida de la aplicación de Kubernetes

Azure Dev Spaces proporciona una experiencia de desarrollo de Kubernetes rápida e iterativa para los equipos. Con una configuración de máquina de desarrollo mínima, puede ejecutar y depurar contenedores de forma iterativa directamente en Azure Kubernetes Service (AKS). Puede desarrollar en Windows, Mac o Linux mediante herramientas familiares como Visual Studio, Visual Studio Code o la línea de comandos.

Como ya se ha mencionado, Azure Dev Spaces usa gráficos de Helm al implementar aplicaciones basadas en contenedores.

Azure Dev Spaces ayuda a los equipos de desarrollo a ser más productivos en Kubernetes porque permite iterar con rapidez y depurar código directamente en un clúster de Kubernetes global en Azure simplemente mediante el uso de Visual Studio 2017 o Visual Studio Code. Ese clúster de Kubernetes en Azure es un clúster de Kubernetes administrado compartido, por lo que su equipo puede colaborar. Puede desarrollar código de forma aislada, después implementarlo en el clúster global y realizar pruebas de un extremo a otro con otros componentes sin tener que replicar ni simular dependencias.

Como se muestra en la figura 4-9, la característica distintiva de Azure Dev Spaces es la capacidad de crear "espacios" que se pueden ejecutar integrados con el resto de la implementación global en el clúster:

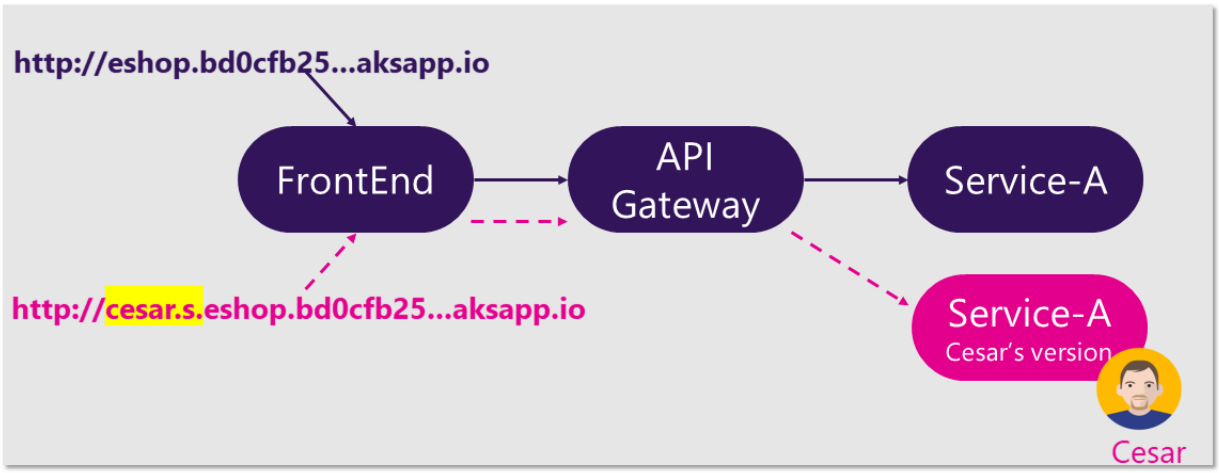


Figura 4-9. Uso de varios espacios en Azure Dev Spaces

Azure Dev Spaces puede mezclar y combinar de forma transparente los microservicios de producción con la instancia de contenedor de desarrollo para facilitar las pruebas de nuevas versiones. Básicamente, puede configurar un espacio de desarrollo compartido en Azure. Así, cada programador se puede centrar exclusivamente en su parte de la aplicación y desarrollar de forma iterativa código previo a la confirmación en un espacio de desarrollo que ya contenga todos los demás servicios y recursos en la nube de los que sus escenarios dependen. Las dependencias siempre estarán actualizadas y los desarrolladores trabajarán de una manera que se asemeja bastante a un entorno de producción.

En Azure Dev Spaces existe el concepto de espacio, que permite trabajar de manera aislada y sin riesgo de romper el código de los miembros del equipo. Esta característica se basa en prefijos de dirección URL. Si usa un prefijo de espacio de desarrollo en la dirección URL para la solicitud de un contenedor, Azure Dev Spaces ejecutará una versión especial del contenedor que se implementa para ese espacio, si existe. En caso contrario, se ejecutará la versión global o consolidada.

Para obtener un ejemplo concreto, vea la página wiki de eShopOnContainers en Azure Dev Spaces.

Para obtener más información, vea Desarrollo en equipo con Azure Dev Spaces.

Recursos adicionales

- **Guía de inicio rápido: Implementación de un clúster de Azure Kubernetes Service (AKS)**
  /azure/aks/kubernetes-walkthrough-portal
- **Azure Dev Spaces**
  /azure/dev-spaces/azure-dev-spaces
- **Kubernetes.** Sitio oficial.
  https://kubernetes.io/

Uso de Azure Service Fabric

Azure Service Fabric surgió de la transición que Microsoft realizó al dejar de ofrecer productos empaquetados, que normalmente tenían un estilo monolítico, para ofrecer servicios. La experiencia de crear y usar servicios de gran tamaño a escala, como Azure SQL Database, Azure Cosmos DB, Azure Service Bus o Backend de Cortana, definió el formato de Service Fabric. La plataforma evolucionó con el tiempo, a medida que la adoptaron cada vez más servicios. Cabe destacar que Service Fabric tuvo que ejecutarse no solo en Azure sino también en implementaciones independientes de Windows Server.

El objetivo de Service Fabric es solucionar los arduos problemas de compilar y ejecutar un servicio y usar recursos de infraestructura de forma eficaz, de manera que los equipos puedan resolver problemas empresariales con un enfoque de microservicios.

Service Fabric proporciona dos áreas generales para ayudarlo a crear aplicaciones que usan un enfoque de microservicios:

- Una plataforma que proporciona servicios de sistema para implementar, escalar, actualizar, detectar y reiniciar servicios erróneos, detectar la ubicación del servicio, administrar el estado y supervisar el mantenimiento. Estos servicios de sistema efectivamente habilitan muchas de las características de microservicios descritas anteriormente.
- Programar API, o marcos, para ayudarlo a crear aplicaciones como microservicios: actores fiables y servicios de confianza. Puede elegir cualquier código para generar el microservicio, pero estas API facilitan el trabajo y se integran con la plataforma a un nivel más profundo. De este modo, puede obtener información de mantenimiento y diagnóstico o puede sacar partido de la administración de estado confiable.

Para crear su servicio puede usar cualquier tecnología ya que Service Fabric no interviene en el proceso. Sin embargo, proporciona API de programación integradas que facilitan la creación de microservicios.

Como se muestra en la figura 4-10, puede crear y ejecutar microservicios en Service Fabric como procesos simples o como contenedores de Docker. También es posible mezclar microservicios basados en contenedores con microservicios basados en procesos en el mismo clúster de Service Fabric.

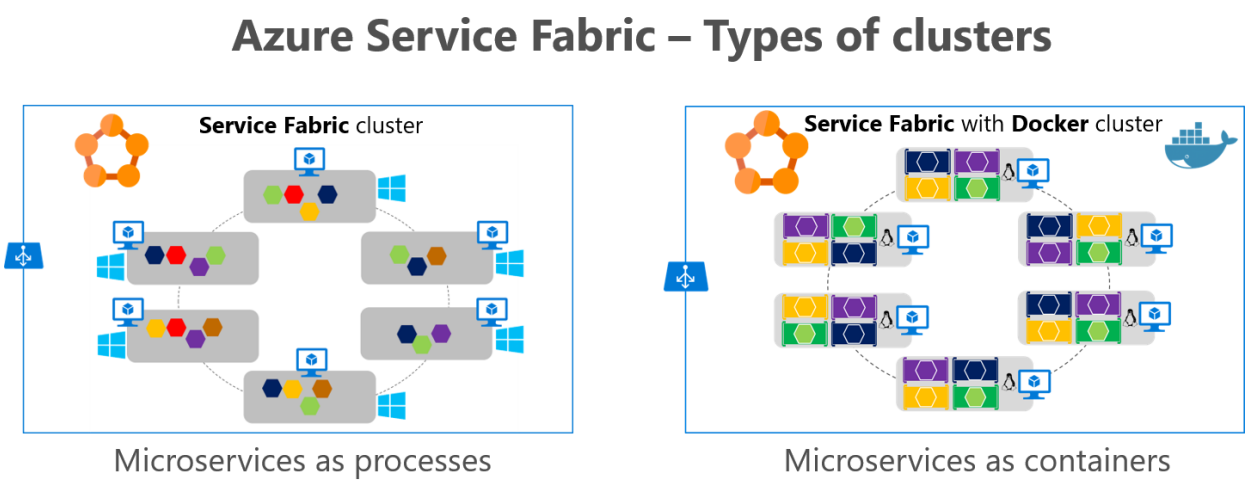


Figura 4-10. Implementar microservicios como procesos o como contenedores en Azure Service Fabric

En la primera imagen, verá microservicios como procesos, donde cada nodo ejecuta un proceso para cada microservicio. En la segunda imagen, verá microservicios como contenedores, donde cada nodo ejecuta Docker con varios contenedores, un contenedor por microservicio. Los clústeres de Service Fabric basados en hosts de Linux y Windows pueden ejecutar contenedores de Docker Linux y Windows, respectivamente.

Para más información actualizada sobre la compatibilidad con contenedores en Azure Service Fabric, vea Service Fabric y contenedores.

Service Fabric es un buen ejemplo de una plataforma en la que puede definir una arquitectura lógica diferente (microservicios empresariales o contextos limitados) de la implementación física. Por ejemplo, si implementa en Azure Service Fabric los servicios de confianza con estado, que se describen más adelante en la sección Microservicios sin estado frente a microservicios con estado, puede tener un concepto de microservicio empresarial con varios servicios físicos.

Como se muestra en la figura 4-10, y bajo una perspectiva de microservicio lógico o empresarial, al implementar un servicio de confianza con estado de Service Fabric, normalmente deberá implementar dos niveles de servicios. El primero es el servicio de confianza con estado back-end, que administra varias particiones (cada partición es un servicio con estado). El segundo es el servicio front-end, o servicio de puerta de enlace, que se encarga del enrutamiento y de la agregación de datos en varias particiones o instancias de

servicio con estado. Este servicio de puerta de enlace también controla la comunicación del lado cliente con los bucles de reintento que acceden al servicio back-end. Se denomina servicio de puerta de enlace si implementa el servicio personalizado, pero alternativamente también puede usar el proxy inverso estándar de Service Fabric.

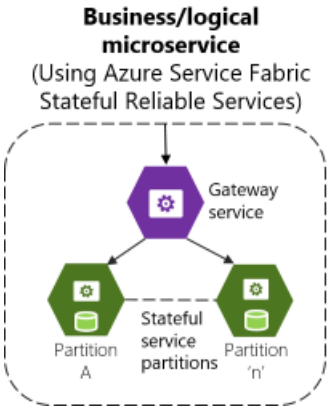


Figura 4-11. Microservicio de negocio con varias instancias de servicio con estado y un front-end con puerta de enlace personalizado

En cualquier caso, al usar servicios de confianza con estado de Service Fabric, también dispone de un microservicio lógico o empresarial (contexto limitado) que se compone de varios servicios físicos. Cada uno de ellos, el servicio de puerta de enlace y el servicio de partición podrían implementarse como servicios ASP.NET Web API, como se muestra en la figura 4-11. Service Fabric está recomendado para admitir varios servicios de confianza con estado en contenedores.

En Service Fabric, puede agrupar e implementar grupos de servicios como una aplicación de Service Fabric, que es la unidad de empaquetado e implementación para el orquestador o clúster. Por tanto, la aplicación de Service Fabric también podría asignarse a este límite de microservicio o contexto limitado lógico y empresarial autónomo, por lo que podría implementar estos servicios de forma autónoma.

Service Fabric y contenedores

Con respecto a los contenedores de Service Fabric, también puede implementar servicios en las imágenes de contenedor dentro de un clúster de Service Fabric. Como se muestra en la figura 4-12, la mayoría de las veces solo habrá un contenedor por cada servicio.

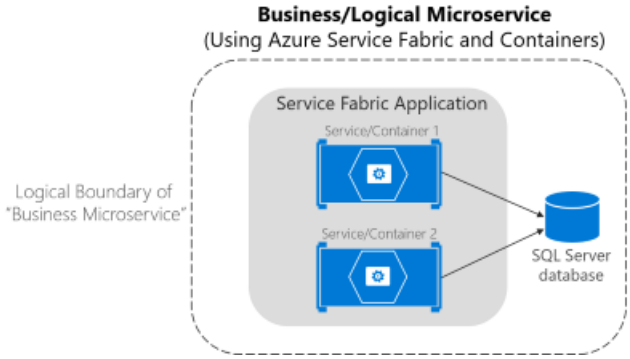


Figura 4-12. Microservicio empresarial con varios servicios (contenedores) en Service Fabric

Una aplicación de Service Fabric puede ejecutar varios contenedores que acceden a una base de datos externa y el conjunto completo sería el límite lógico de un microservicio empresarial. Pero los contenedores llamados "asociados" (dos contenedores que deben implementarse conjuntamente como parte de un servicio lógico) también son posibles en Service Fabric. Lo importante es que un microservicio empresarial sea el límite lógico alrededor de varios elementos cohesivos. En muchos casos, puede que sea un único servicio con un solo modelo de datos, pero en otros casos también es posible que tenga varios servicios físicos.

Tenga en cuenta que se pueden combinar servicios en procesos y servicios en contenedores en la misma aplicación de Service Fabric, como se muestra en la figura 4-13.

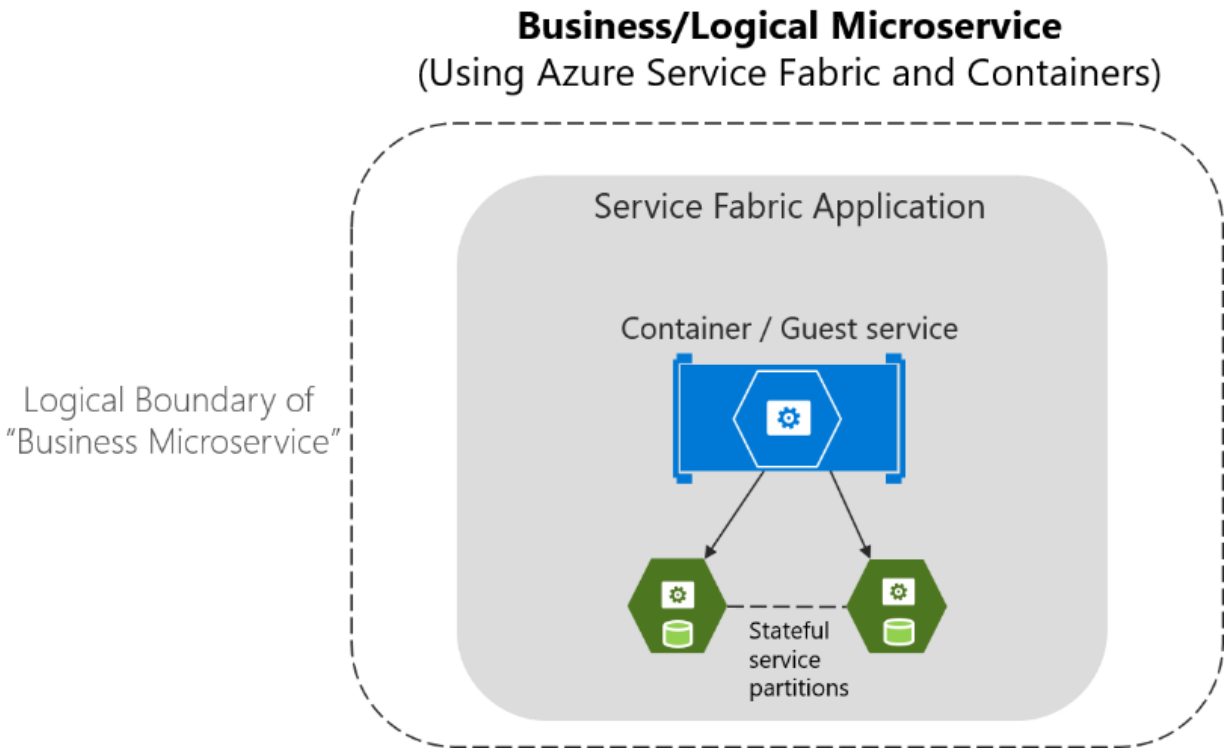


Figura 4-13. Microservicio empresarial asignado a una aplicación de Service Fabric con contenedores y servicios con estado

Para más información sobre la compatibilidad de contenedor en Azure Service Fabric, vea Service Fabric y contenedores.

Microservicios sin estado frente a microservicios con estado

Como se ha mencionado anteriormente, cada microservicio (contexto limitado lógico) debe tener su modelo de dominio (datos y lógica). En el caso de los microservicios sin estado, las bases de datos serán externas y se usarán opciones relacionales, como SQL Server, u opciones de NoSQL, como MongoDB o Azure Cosmos DB.

Pero los propios servicios también pueden ser con estado en Service Fabric, lo que significa que los datos se encuentran en el microservicio. Estos datos pueden existir no solo en el mismo servidor, sino dentro del proceso del microservicio, en memoria, conservados en discos duros y replicados en otros nodos. En la figura 4-14 se muestran los distintos enfoques.

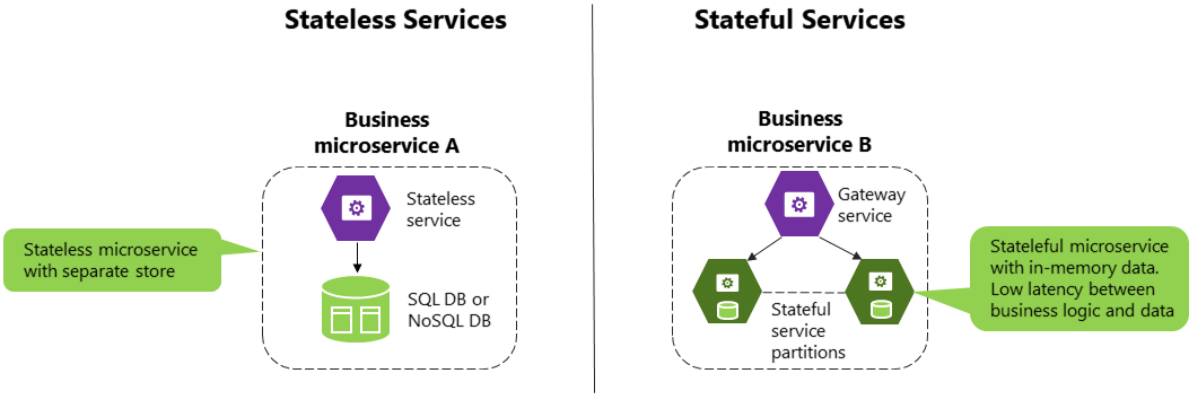


Figura 4-14. Microservicios sin estado frente a microservicios con estado

En los servicios sin estado, el estado (persistencia, base de datos) se mantiene fuera del microservicio. En los servicios con estado, el estado se mantiene dentro del microservicio. Un enfoque sin estado es perfectamente válido y es más fácil de implementar que los microservicios con estado, ya que el enfoque es similar a los

patrones tradicionales y conocidos. Pero los microservicios sin estado imponen latencia entre el proceso y los orígenes de datos. También implican más piezas móviles cuando se intenta mejorar el rendimiento con memoria caché y colas adicionales. El resultado es que puede acabar con arquitecturas complejas que tienen demasiados niveles.

Por el contrario, los microservicios con estado pueden destacar en escenarios avanzados, ya que no hay latencia entre los datos y la lógica del dominio. El procesamiento intenso de datos, los back-ends de juegos, las bases de datos como servicio y otros escenarios con baja latencia se benefician de los servicios con estado, que habilitan el estado local para un acceso más rápido.

Los servicios sin estado y los servicios con estado se complementan. Por ejemplo, como puede ver en el diagrama de la derecha en la figura 4-14, un servicio con estado se puede dividir en varias particiones. Para acceder a esas particiones, es posible que deba contar con un servicio sin estado que actúe como un servicio de puerta de enlace que sepa cómo dirigirse a cada partición en función de las claves de partición.

Los servicios con estado tienen diversos inconvenientes. Imponen un alto nivel de complejidad en el escalado horizontal. Las funciones que normalmente se implementarían mediante sistemas de bases de datos externas se debe abordar en tareas como la replicación de datos a través de microservicios con estado y la creación de particiones de datos. Sin embargo, esta es una de las áreas en que un orquestador como Azure Service Fabric con sus servicios de confianza con estado puede ayudar al máximo, ya que simplifica el desarrollo y el ciclo de vida de los microservicios con estado mediante la API de Reliable Services y Reliable Actors.

Otros marcos de microservicio que permiten los servicios con estado, admiten el patrón de actor y mejoran la tolerancia a errores y la latencia entre la lógica y los datos de negocios son Microsoft Orleans, de Microsoft Research, y Akka.NET. Actualmente, ambos marcos están mejorando su compatibilidad con Docker.

Recuerde que los contenedores de Docker son sin estado. Si quiere implementar un servicio con estado, debe contar con uno de los marcos prescriptivos y de nivel superior adicionales que se han indicado anteriormente.

Uso de Azure Service Fabric Mesh

Azure Service Fabric Mesh es un servicio totalmente administrado que permite a los desarrolladores compilar e implementar aplicaciones críticas sin tener que administrar ninguna infraestructura. Utilice Service Fabric Mesh para compilar y ejecutar aplicaciones de microservicios seguras y distribuidas que se escalan a petición.

Como se muestra en la figura 4-15, las aplicaciones hospedadas en Service Fabric Mesh se ejecutan y escalan sin preocuparse por la infraestructura de la que dependen.

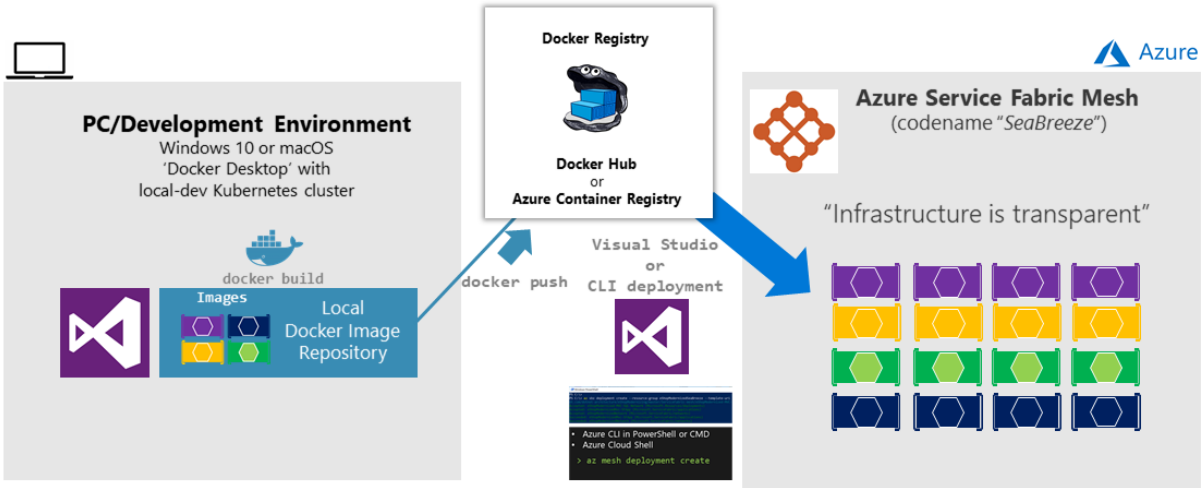


Figura 4-15. Implementación de una aplicación de microservicios/contenedores en Service Fabric Mesh

En segundo plano, Service Fabric Mesh consta de clústeres de miles de máquinas. Todas las operaciones de clúster se ocultan a los desarrolladores. Basta con cargar los contenedores y especificar los recursos necesarios,

los requisitos de disponibilidad y los límites de recursos. Service Fabric Mesh asigna automáticamente la infraestructura solicitada por la implementación de la aplicación y también controla los errores de infraestructura, asegurándose de que las aplicaciones tienen una alta disponibilidad. Solo necesita preocuparse del estado y de la capacidad de respuesta de la aplicación, nunca de la infraestructura.

Para obtener más información, vea la documentación de Service Fabric Mesh.

Elección de orquestadores en Azure

En la siguiente tabla se proporcionan instrucciones sobre qué orquestador se debe usar en función de las cargas de trabajo y el foco del sistema operativo.

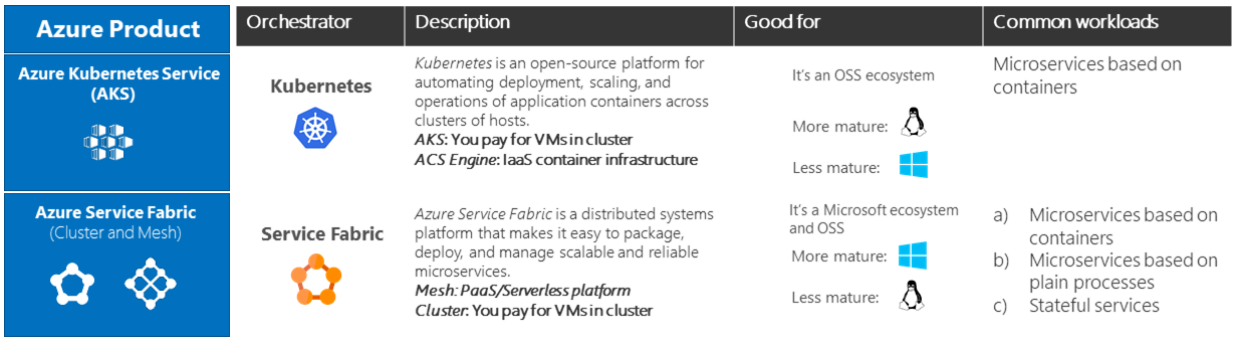


Figura 4-16. Selección del orquestador en instrucciones de Azure

ANTERIOR SIGUIENTE

Implementación en Azure Kubernetes Service (AKS)

06/03/2021 • 2 minutes to read • Edit Online

Puede interactuar con AKS mediante el sistema operativo cliente que prefiera (Windows, macOS o Linux) con la interfaz de la línea de comandos de Azure (CLI de Azure) instalada. Para obtener más detalles, vea la documentación de la CLI de Azure y la guía de instalación de los entornos disponibles.

Creación del entorno de AKS en Azure

Existen varias formas de crear el entorno de AKS. Puede hacerlo mediante comandos de la CLI de Azure o por medio de Azure Portal.

Aquí puede ver algunos ejemplos de uso de la CLI de Azure para crear el clúster y de Azure Portal para revisar los resultados. También deberá tener Kubectl y Docker en el equipo de desarrollo.

Creación del clúster de AKS

Cree el clúster de AKS con este comando (el grupo de recursos debe existir):

```
az aks create --resource-group explore-docker-aks-rg --name explore-docker-aks --node-vm-size Standard_B2s --node-count 1 --generate-ssh-keys --location westeurope
```

NOTE

Los valores de los parámetros `--node-vm-size` y `--node-count` son lo bastante buenos para una aplicación de ejemplo o desarrollo.

Una vez finalizado el trabajo de creación, puede ver:

- El clúster de AKS creado en el grupo de recursos inicial
- Un nuevo grupo de recursos relacionado que contiene los recursos relacionados con el clúster de AKS, como se muestra en las siguientes imágenes.

El grupo de recursos inicial, con el clúster de AKS:

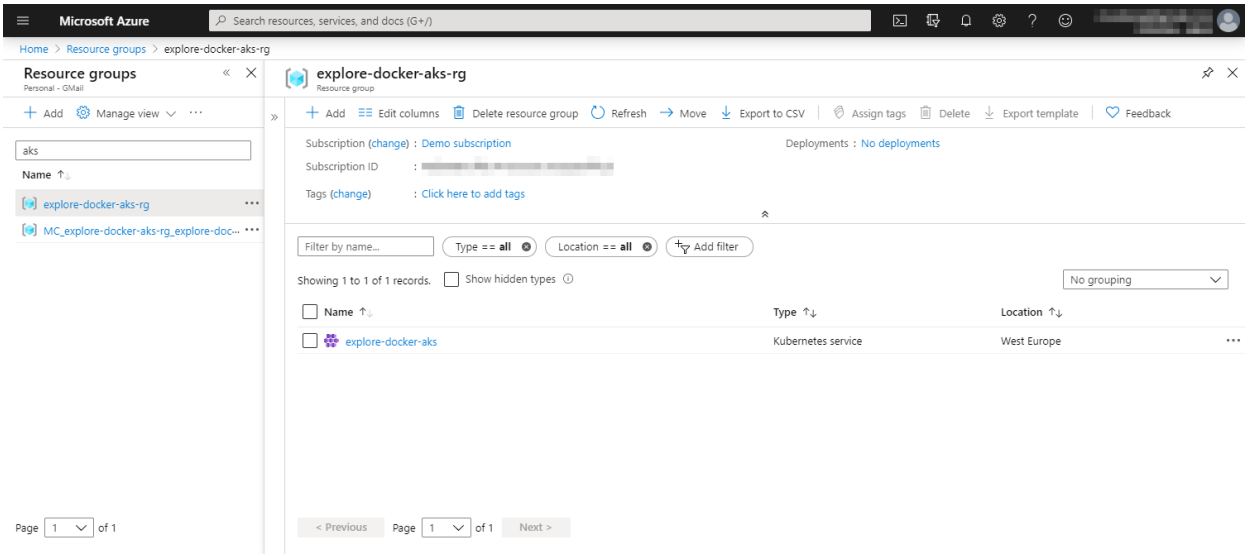


Figura 4-17. Vista de grupo de recursos de AKS en Azure.

El grupo de recursos del clúster de AKS:

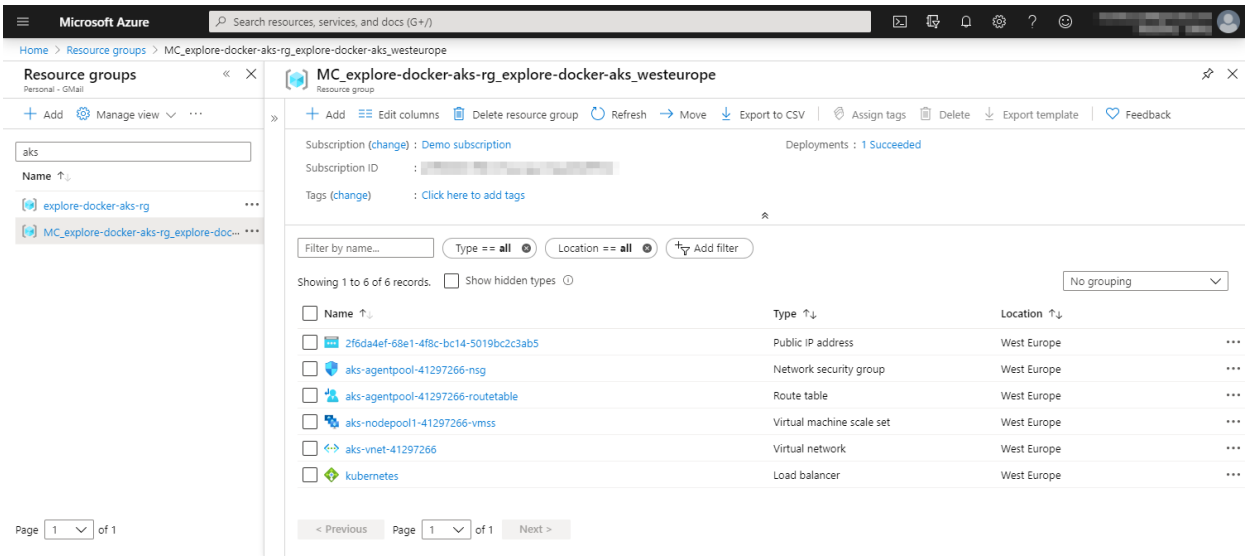


Figura 4-18. Vista de AKS en Azure.

> **IMPORTANT**
>
> En general, no debería tener que modificar los recursos del grupo de recursos del clúster de AKS. Por ejemplo, el grupo de recursos se elimina al eliminar el clúster de AKS.

También puede ver el nodo creado mediante `Azure CLI` y `Kubectl`.

En primer lugar, obtenga las credenciales:

```
az aks get-credentials --resource-group explore-docker-aks-rg --name explore-docker-aks
```

```
miguel@LAPTOP-MV:/mnt/c/Users/Miguel$ az aks get-credentials --resource-group explore-docker-aks-rg --name explore-docker-aks
Merged "explore-docker-aks" as current context in /home/miguel/.kube/config
miguel@LAPTOP-MV:/mnt/c/Users/Miguel$
```

Figura 4-19. Resultado del comando `aks get-credentials`.

Después, obtenga los nodos de Kubectl:

```
kubectl get nodes
```

```
miguel@LAPTOP-MV:/mnt/c/Users/Miguel$ kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-nodepool1-41297266-vmss000000   Ready    agent   36m   v1.15.10
miguel@LAPTOP-MV:/mnt/c/Users/Miguel$
```

Figura 4-20. Resultado del comando `kubectl get nodes`.

Entorno de desarrollo para aplicaciones de Docker

06/03/2021 • 3 minutes to read • Edit Online

Opciones de herramientas de desarrollo: IDE o editor

Con independencia de que prefiera un IDE eficaz y completo, o un editor ligero y ágil, Microsoft le puede ayudar en el desarrollo de aplicaciones de Docker.

Visual Studio Code y la CLI de Docker (herramientas multiplataforma para Mac, Linux y Windows)

Si prefiere un editor ligero y multiplataforma que admita cualquier lenguaje de programación, puede usar Visual Studio Code y la CLI de Docker. Estos productos proporcionan una experiencia sencilla y sólida que es fundamental para agilizar el flujo de trabajo del desarrollador. Al instalar "Docker para Mac" o "Docker para Windows" (entorno de desarrollo), los desarrolladores de Docker pueden usar una sola CLI de Docker para compilar aplicaciones para Windows o Linux (entorno de tiempo de ejecución). Además, Visual Studio Code admite extensiones para Docker con IntelliSense para archivos Dockerfile y tareas de acceso directo para ejecutar comandos de Docker desde el editor.

> **NOTE**
>
> Para descargar Visual Studio Code, vaya a https://code.visualstudio.com/download.
>
> Para descargar Docker para Mac y Windows, vaya a https://www.docker.com/products/docker.

Visual Studio con herramientas de Docker (equipo de desarrollo de Windows)

Se recomienda usar Visual Studio 2019 con las herramientas de Docker integradas habilitadas. Con Visual Studio, puede desarrollar, ejecutar y validar las aplicaciones directamente en el entorno de Docker seleccionado. Presione F5 para depurar la aplicación (de un solo contenedor o de varios contenedores) directamente en un host de Docker, o bien presione CTRL+F5 para editar y actualizar la aplicación sin tener que volver a compilar el contenedor. Esta es la opción más sencilla y más eficaz para los desarrolladores de Windows que crean contenedores de Docker para Linux o Windows.

Visual Studio para Mac (equipo de desarrollo de Mac)

Puede usar Visual Studio para Mac al desarrollar aplicaciones basadas en Docker. Visual Studio para Mac ofrece un IDE más completo en comparación con Visual Studio Code para Mac.

Opciones de lenguaje y marco

Puede desarrollar aplicaciones de Docker mediante el uso de herramientas de Microsoft con los lenguajes más modernos. A continuación se muestra una lista inicial, aunque no está limitado a ella:

- .NET y ASP.NET Core
- Node.js
- Ir
- Java
- Ruby
- Python

Básicamente, puede usar cualquier lenguaje moderno compatible con Docker en Linux o Windows.

ANTERIOR SIGUIENTE

Flujo de trabajo de desarrollo de bucle interior para aplicaciones de Docker

06/03/2021 • 24 minutes to read • Edit Online

Antes de desencadenar el flujo de trabajo de bucle exterior que comprende el ciclo completo de DevOps, todo comienza en la máquina de cada desarrollador, al codificar la propia aplicación, utilizar sus lenguajes o plataformas preferidos y probarla localmente (figura 4-21). Pero en todos los casos, tendrá un aspecto importante en común, con independencia del lenguaje, el marco o las plataformas que elija. En este flujo de trabajo concreto, los contenedores de Docker siempre se desarrollan y se prueban en este entorno y no otro, pero en local.

Figura 4-21. Contexto de desarrollo de bucle interno

El contenedor o la instancia de una imagen de Docker contiene estos componentes:

- Una selección de sistema operativo (por ejemplo, una distribución de Linux o Windows)
- Archivos agregados por el desarrollador (por ejemplo, binarios de aplicación)
- Información de configuración (por ejemplo, configuración de entorno y dependencias)
- Instrucciones sobre los procesos que debe ejecutar Docker

Puede configurar el flujo de trabajo de desarrollo de bucle interno que usa Docker como proceso (lo que se describe en la sección siguiente). Tenga en cuenta que no se incluyen los pasos iniciales para configurar el entorno, ya que basta con hacerlo una vez.

Compilación de una sola aplicación en un contenedor de Docker con Visual Studio Code y la CLI de Docker

Las aplicaciones se componen de sus propios servicios, además de bibliotecas adicionales (dependencias).

La figura 4-22 muestra los pasos básicos que normalmente hay que llevar a cabo para compilar una aplicación de Docker, seguidos de una descripción detallada de cada paso.

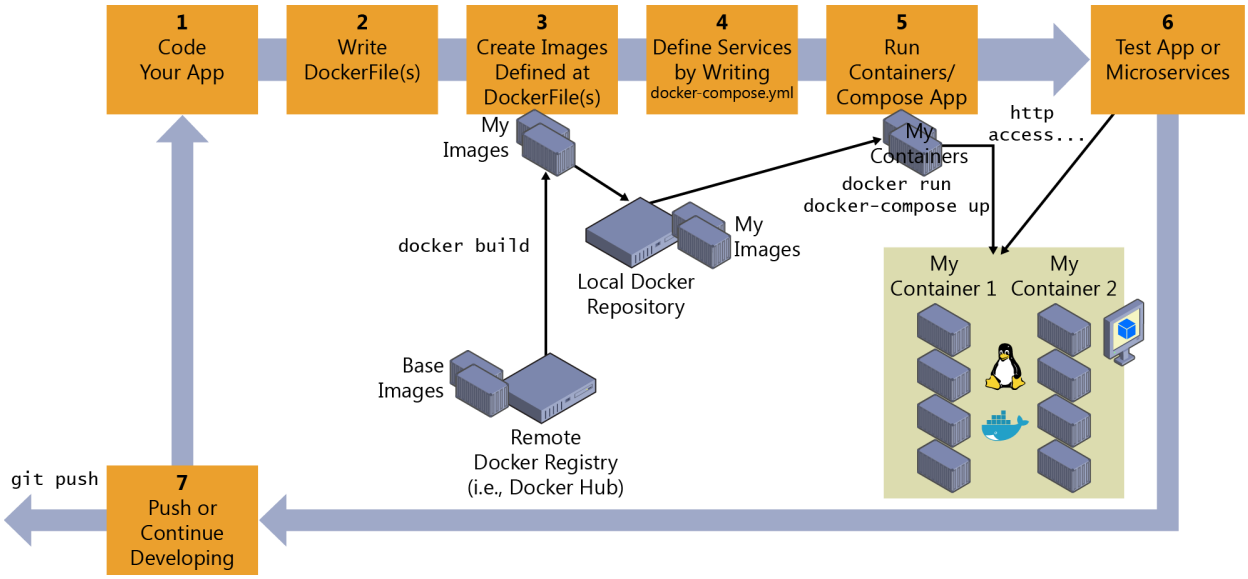


Figura 4-22. Flujo de trabajo general del ciclo de vida de aplicaciones en contenedor con la CLI de Docker

Paso 1: Inicio de la codificación en Visual Studio Code y creación de la instantánea inicial de la aplicación o el servicio

La forma de desarrollar una aplicación es similar a la forma en que se lleva a cabo sin Docker. La diferencia es que durante el desarrollo, la aplicación o los servicios que se están implementando y probando se ejecutan en contenedores de Docker situados en el entorno local (como una VM Linux o Windows).

Configuración del entorno local

Con las últimas versiones de Docker Desktop para Mac y Windows, desarrollar aplicaciones de Docker es más fácil que nunca y la configuración es sencilla.

> **TIP**
>
> Para obtener instrucciones sobre cómo configurar Docker Desktop para Windows, vaya a https://docs.docker.com/docker-for-windows/.
>
> Para obtener instrucciones sobre cómo configurar Docker Desktop para Mac, vaya a https://docs.docker.com/docker-for-mac/.

Además, necesitará un editor de código para que realmente pueda desarrollar su aplicación al tiempo que usa la CLI de Docker.

Microsoft ofrece Visual Studio Code, que es un editor de código ligero compatible con Windows, Linux y macOS y que proporciona IntelliSense con compatibilidad con numerosos lenguajes (JavaScript,. NET, Go, Java, Ruby, Python y la mayoría de lenguajes modernos), depuración, integración con Git y compatibilidad con extensiones. Este editor es muy adecuado para desarrolladores de macOS y Linux. En Windows, también puede usar Visual Studio.

> **TIP**
>
> Para instrucciones sobre cómo instalar Visual Studio Code para Windows, Linux o macOS, vaya a https://code.visualstudio.com/docs/setup/setup-overview/.
>
> Para obtener instrucciones sobre cómo configurar Docker para Mac, vaya a https://docs.docker.com/docker-for-mac/.

Puede trabajar con la CLI de Docker y escribir el código con cualquier editor de código, pero el uso de Visual Studio Code con la extensión de Docker facilita la creación de archivos `Dockerfile` y `docker-compose.yml` en el área de trabajo. También puede ejecutar tareas y scripts en el IDE de Visual Studio Code que ejecuten

comandos de Docker con la CLI de Docker que se encuentra debajo.

La extensión de Docker para VS Code ofrece las siguientes características:

- Generación automática de archivos `Dockerfile` y `docker-compose.yml`
- Sugerencias de mantenimiento del puntero y resaltado de sintaxis en archivos `docker-compose.yml` y `Dockerfile`
- IntelliSense (finalizaciones) para archivos `Dockerfile` y `docker-compose.yml`
- Linting (errores y advertencias) en archivos `Dockerfile`
- Integración con la paleta de comandos (F1) para los comandos de Docker más comunes
- Integración con el explorador para administrar imágenes y contenedores
- Implementación de imágenes de DockerHub y de instancias de Azure Container Registry en Azure App Service

Para instalar la extensión de Docker, presione Ctrl+Mayús+P, escriba `ext install` y ejecute el comando Instalar extensión para que aparezca la lista de extensiones de Marketplace. A continuación, escriba **docker** para filtrar los resultados y luego seleccione la extensión de compatibilidad con Docker, tal y como se muestra en la figura 4-23.

Figura 4-23. Instalación de la extensión de Docker para Visual Studio Code

Paso 2: Creación de un DockerFile relacionado con una imagen existente (entornos de desarrollo o sistemas operativos estándar como .NET, Node.js y Ruby)

Necesitará un `DockerFile` por imagen personalizada que quiera crear y por contenedor que quiera implementar. Si la aplicación se compone de un único servicio personalizado, necesita un solo `DockerFile`. Pero si la aplicación se compone de varios servicios (como en una arquitectura de microservicios), necesitará un `Dockerfile` por servicio.

El archivo `DockerFile` normalmente se coloca en la carpeta raíz de la aplicación o el servicio y contiene los comandos necesarios para que Docker sepa cómo configurar y ejecutar esa aplicación o ese servicio. Puede crear el elemento `DockerFile` y agregarlo al proyecto junto con el código (node.js, .NET, etc.) o, si no está familiarizado con el entorno, eche un vistazo a la siguiente sugerencia.

> **TIP**
> Puede usar la extensión de Docker como guía al usar los archivos `Dockerfile` y `docker-compose.yml` relacionados con los contenedores de Docker. Con el tiempo, es probable que escriba este tipo de archivos sin esta herramienta, pero el uso de la extensión de Docker es un buen punto de partida que acelerará su curva de aprendizaje.

En la figura 4-24 puede ver los pasos necesarios para agregar archivos de Docker a un proyecto con la extensión de Docker para VS Code:

1. Abra la paleta de comandos, escriba "**docker**" y seleccione "**Add Docker Files to Workspace**" (Agregar archivos de Docker al área de trabajo).
2. Selección de la plataforma de aplicación (ASP.NET Core)

3. Selección del sistema operativo (Linux)
4. Inclusión de archivos de Docker Compose opcionales
5. Especificación de los puertos para publicar (80, 443)
6. Seleccionar el proyecto

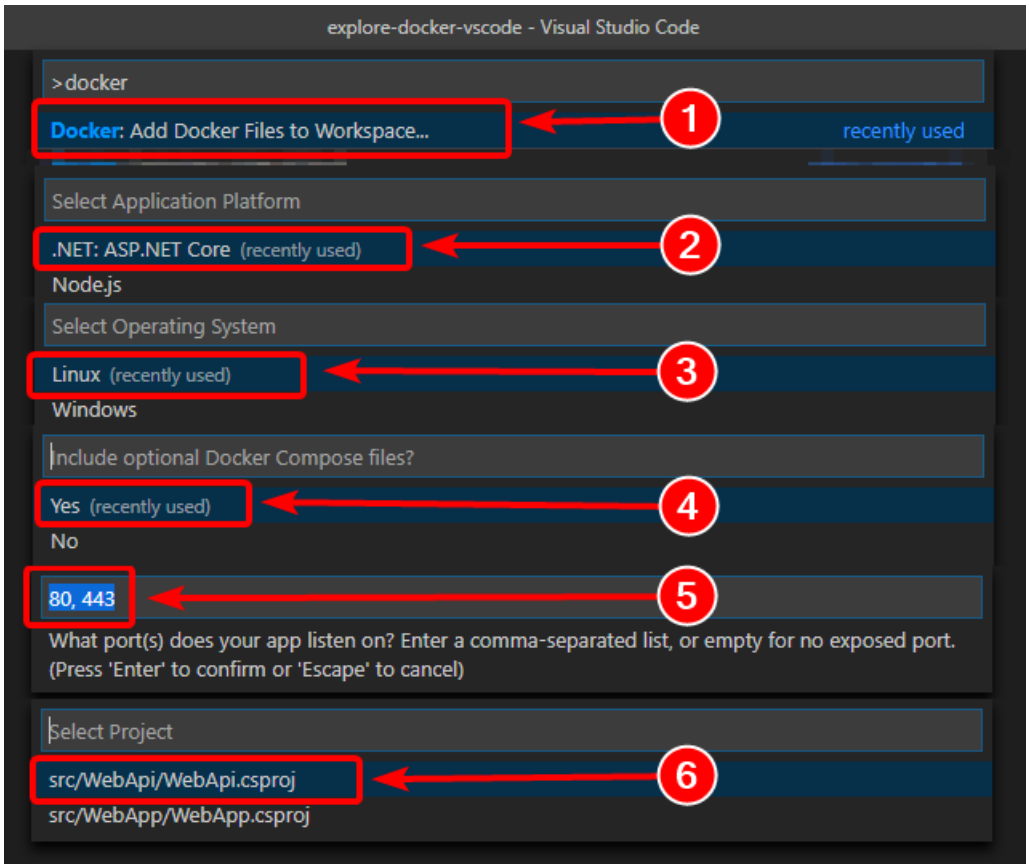


Figura 4-24. Archivos de Docker agregados con el comando **Add Docker files to Workspace** (Agregar archivos de Docker al área de trabajo)

Cuando agregue un documento DockerFile, especifique la imagen base de Docker que va a usar (como el uso de `FROM mcr.microsoft.com/dotnet/aspnet`). Normalmente, creará la imagen personalizada sobre una imagen base que obtendrá de cualquier repositorio oficial en el registro de Docker Hub (como una imagen para .NET o la correspondiente para Node.js).

> **TIP**
> Tiene que repetir este procedimiento para cada proyecto de la aplicación, aunque la extensión le pide que sobrescriba el archivo de Docker Compose generado después de la primera vez. Debe responder que no quiere sobrescribirlo, para que la extensión cree archivos independientes de Docker Compose que después pueda combinar manualmente, antes de ejecutar Docker Compose.

Uso de una imagen oficial de Docker existente

El uso de un repositorio oficial de una pila de lenguaje con un número de versión garantiza que haya las mismas características de lenguaje disponibles en todas las máquinas (incluidas las de desarrollo, pruebas y producción).

Este es un DockerFile de ejemplo para un contenedor de .NET:

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["src/WebApi/WebApi.csproj", "src/WebApi/"]
RUN dotnet restore "src/WebApi/WebApi.csproj"
COPY . .
WORKDIR "/src/src/WebApi"
RUN dotnet build "WebApi.csproj" -c Release -o /app/build

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

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

En este caso, la imagen se basa en la versión 5.0 de la imagen oficial de Docker para ASP.NET Core (multiarquitectura de Linux y Windows), según la línea `FROM mcr.microsoft.com/dotnet/aspnet:5.0`. (Para obtener más información sobre este tema, vea las páginas sobre la imagen de Docker de ASP.NET Core y la imagen de Docker de .NET).

En el Dockerfile, también puede indicar a Docker que escuche el puerto TCP que se va a usar en tiempo de ejecución (por ejemplo, el puerto 80 o 443).

Puede especificar otros valores de configuración en el Dockerfile, según el lenguaje y el marco que use. Por ejemplo, la línea `ENTRYPOINT` con `["dotnet", "WebMvcApplication.dll"]` indica a Docker que ejecute una aplicación de .NET. Si usa el SDK y la CLI de .NET (`dotnet CLI`) para compilar y ejecutar la aplicación de .NET, este valor sería diferente. El punto clave aquí es que la línea ENTRYPOINT y otros valores varían según el lenguaje y la plataforma que se elijan para la aplicación.

> **TIP**
>
> Para obtener más información sobre cómo crear imágenes de Docker para aplicaciones de .NET, vaya a /dotnet/core/docker/building-net-docker-images.
>
> Para más información sobre cómo crear sus propias imágenes, vaya a https://docs.docker.com/engine/tutorials/dockerimages/.

Uso de repositorios de imágenes multiarquitectura

Un solo nombre de imagen en un repositorio puede contener variantes de plataforma, como una imagen de Linux y una imagen de Windows. Esta característica permite a los proveedores como Microsoft (creadores de imágenes base) crear un único repositorio que cubra varias plataformas (es decir, Linux y Windows). Por ejemplo, el repositorio dotnet/aspnet, disponible en el registro de Docker Hub, proporciona compatibilidad con Linux y Windows Nano Server mediante el mismo nombre de imagen.

Al extraer la imagen dotnet/aspnet de un host de Windows, se extrae la variante de Windows, y al extraer el mismo nombre de imagen de un host de Linux, se extrae la variante de Linux.

Creación de la imagen base desde cero

Puede crear su propia imagen de base de Docker desde cero, tal y como se explica en este artículo de Docker. Probablemente este escenario no sea el más adecuado para usuarios que acaban de iniciarse en Docker, pero si quiere establecer los bits específicos de su propia imagen base, puede hacerlo.

Paso 3: Creación de imágenes personalizadas de Docker insertando su servicio en ellas

Por cada servicio personalizado que incluya su aplicación, deberá crear una imagen relacionada. Si la aplicación consta de un único servicio o aplicación web, solo necesitará una imagen.

> **NOTE**
>
> Si se tiene en cuenta el "flujo de trabajo de bucle exterior de DevOps", las imágenes se crearán mediante un proceso de compilación automatizado cada vez que el código fuente se inserte en un repositorio de Git (integración continua), por lo que las imágenes se crearán en ese entorno global a partir de su código fuente.
>
> Pero antes de contemplar la posibilidad de ir a esa ruta de bucle externo, tiene que asegurarse de que la aplicación de Docker funciona correctamente para que no inserte código que pudiera no funcionar correctamente en el sistema de control de código fuente (Git, etc.).
>
> Por lo tanto, cada desarrollador debe realizar en primer lugar todo el proceso de bucle interno para probarlo localmente y continuar desarrollando hasta que quiera insertar una característica completa o cambiarla al sistema de control de código fuente.

Para crear una imagen en el entorno local y usar el Dockerfile, puede emplear el comando build de Docker, como se muestra en la figura 4-25, porque ya etiqueta la imagen automáticamente y compila las imágenes para todos los servicios de la aplicación con un comando simple.

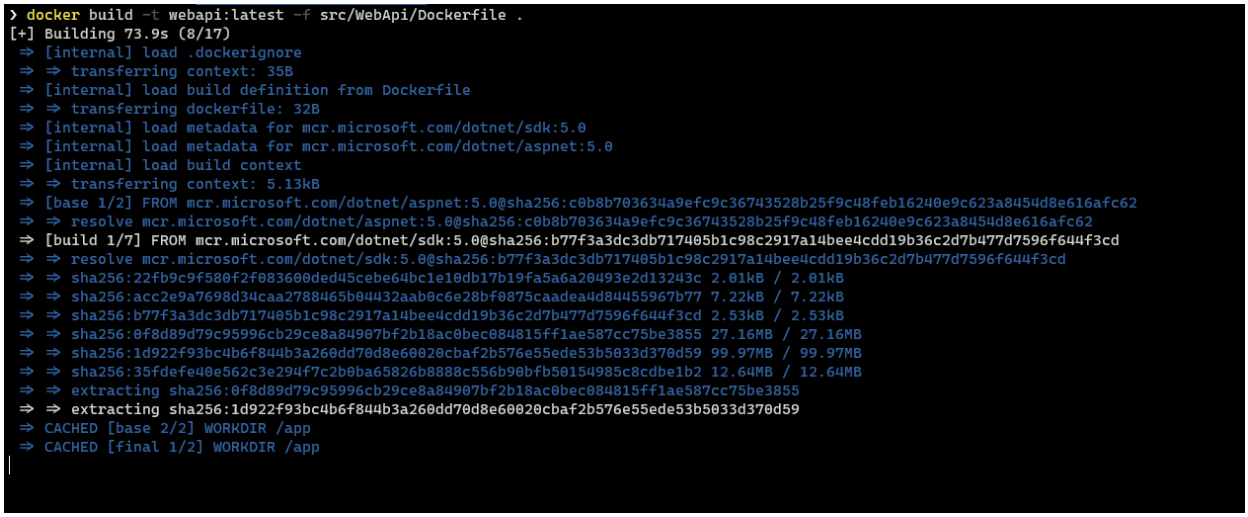


Figura 4-25. Ejecución de la compilación de Docker

Si lo prefiere, en lugar de ejecutar directamente `docker build` desde la carpeta del proyecto, puede generar primero una carpeta implementable con las bibliotecas.NET necesarias mediante la ejecución del comando `dotnet publish` y la posterior ejecución de `docker build`.

En este ejemplo se crea una imagen de Docker de nombre `webapi:latest` (`:latest` es una etiqueta, como una versión específica). Puede seguir este paso para cada imagen personalizada que tenga que crear para la aplicación de Docker compuesta con varios contenedores. Pero en la siguiente sección va a ver que es más fácil hacerlo mediante `docker-compose`.

Puede encontrar las imágenes existentes en el repositorio local (la máquina de desarrollo) mediante el comando `docker images`, como se muestra en la figura 4-26.

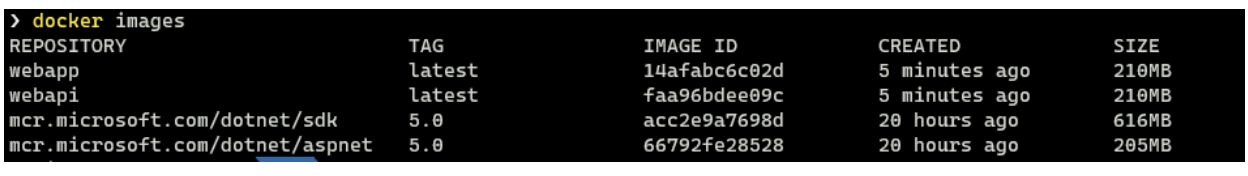


Figura 4-26. Visualización de imágenes existentes con docker images

Paso 4: Definición de los servicios en docker-compose.yml al compilar una aplicación de Docker compuesta de varios contenedores

Con el archivo `docker-compose.yml`, puede definir un conjunto de servicios relacionados para implementarlos como una aplicación compuesta con los comandos de implementación que se explican en la sección siguiente.

Cree ese archivo en la carpeta principal o raíz de su solución; debe tener un contenido similar al que se muestra en este archivo `docker-compose.yml`:

```
version: "3.4"

services:
  webapi:
    image: webapi
    build:
      context: .
      dockerfile: src/WebApi/Dockerfile
    ports:
      - 51080:80
    depends_on:
      - redis
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80

  webapp:
    image: webapp
    build:
      context: .
      dockerfile: src/WebApp/Dockerfile
    ports:
      - 50080:80
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80
      - WebApiBaseAddress=http://webapi

  redis:
    image: redis
```

En este caso concreto, este archivo define tres servicios: el servicio API web (el servicio personalizado), una aplicación web y el servicio Redis (un popular servicio de caché). Cada servicio se implementa como un contenedor, por lo que debe usar una imagen de Docker concreta para cada uno. En el caso de esta aplicación concreta:

- El servicio API web se compila a partir del Dockerfile del directorio `src/WebApi/Dockerfile`.
- El puerto de host 51080 se reenvía al puerto 80 expuesto en el contenedor `webapi`.
- El servicio API web depende del servicio Redis
- La aplicación web accede al servicio API web mediante la dirección interna: `http://webapi`.
- El servicio Redis usa la imagen pública más reciente de Redis extraída del registro de Docker Hub. Redis es un popular sistema de caché para aplicaciones de servidor.

Paso 5: Compilación y ejecución de la aplicación de Docker

Si la aplicación tiene un solo contenedor, para ejecutarla tan solo tiene que implementarla en el host de Docker (máquina virtual o servidor físico). Aunque si la aplicación se compone de varios servicios también tendrá que *componerla*. Veamos las distintas opciones.

Opción A: Ejecución de un único contenedor o servicio

Puede ejecutar la imagen de Docker mediante el comando docker run, tal y como se muestra aquí:

```
docker run -t -d -p 50080:80 webapp:latest
```

En el caso de esta implementación en particular, las solicitudes enviadas al puerto 50080 en el host se van a redirigir al puerto interno 80.

Opción B: Redacción y ejecución de una aplicación de varios contenedores

En la mayoría de los escenarios empresariales, una aplicación de Docker se compone de varios servicios. En estos casos, puede ejecutar el comando `docker-compose up` (figura 4-27), que va a usar el archivo docker-compose.yml creado anteriormente. Al ejecutar este comando se compilan todas las imágenes personalizadas y se implementa la aplicación compuesta con todos sus contenedores relacionados.

```
> docker-compose up
Creating explore-docker-vscode_webapi_1  ... done
Creating explore-docker-vscode_webapp_1  ... done
Attaching to explore-docker-vscode_webapp_1, explore-docker-vscode_webapi_1
webapp_1  | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
webapp_1  |       Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be p
ected data will be unavailable when container is destroyed.
webapp_1  | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
webapp_1  |       No XML encryptor configured. Key {19eedc7c-d3d0-4443-af35-93d4d078119c} may be pe

webapi_1  | info: Microsoft.Hosting.Lifetime[0]
webapi_1  |       Now listening on: http://[::]:80
webapi_1  | info: Microsoft.Hosting.Lifetime[0]
webapi_1  |       Application started. Press Ctrl+C to shut down.
webapi_1  | info: Microsoft.Hosting.Lifetime[0]
webapi_1  |       Hosting environment: Development
webapi_1  | info: Microsoft.Hosting.Lifetime[0]
webapi_1  |       Content root path: /app
webapp_1  | info: Microsoft.Hosting.Lifetime[0]
webapp_1  |       Now listening on: http://[::]:80
webapp_1  | info: Microsoft.Hosting.Lifetime[0]
webapp_1  |       Application started. Press Ctrl+C to shut down.
webapp_1  | info: Microsoft.Hosting.Lifetime[0]
webapp_1  |       Hosting environment: Development
webapp_1  | info: Microsoft.Hosting.Lifetime[0]
webapp_1  |       Content root path: /app
```

Figura 4-27. Resultados de la ejecución del comando "docker-compose up"

Después de ejecutar `docker-compose up`, se implementa la aplicación y sus contenedores relacionados en el host de Docker, tal y como se muestra en la figura 4-28, en la representación de la máquina virtual.

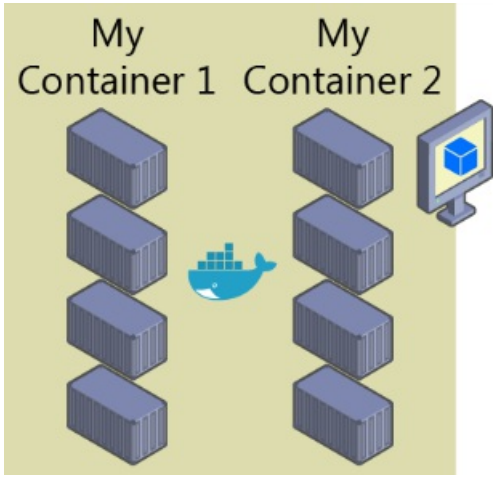


Figura 4-28. Máquina virtual con contenedores de Docker implementados

Paso 6: Prueba de la aplicación de Docker (localmente, en la máquina virtual de CD local)

Este paso varía en función de lo que haga la aplicación.

En "Hola mundo", una API web sencilla de .NET que se implementa como contenedor o servicio único, solo tiene que acceder al servicio facilitando el puerto TCP especificado en el DockerFile.

En el host de Docker, abra un explorador y navegue a ese sitio; debe ver la aplicación o el servicio en ejecución, tal y como se muestra en la figura 4-29.

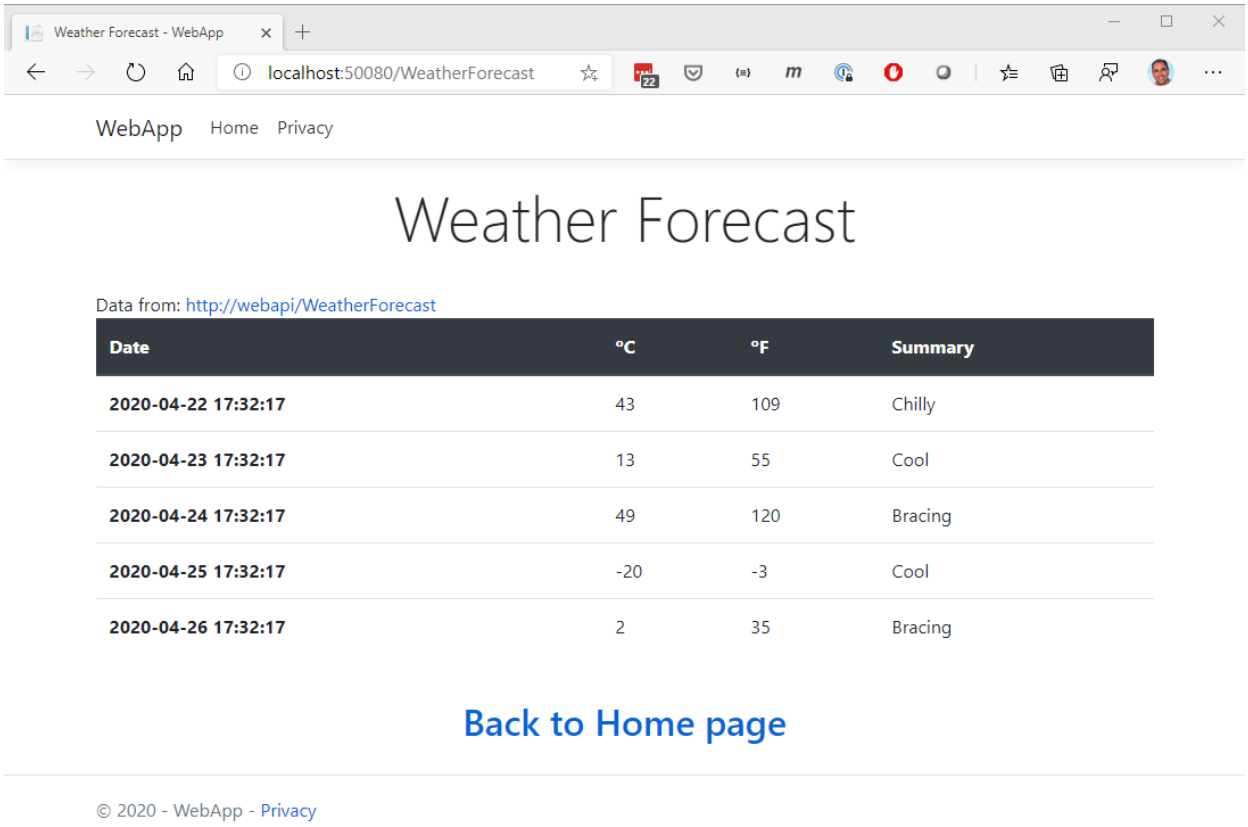


Figura 4-29. Prueba de la aplicación de Docker en local mediante el explorador

Observe que usa el puerto 50080, pero internamente se redirige al puerto 80, porque así es como se han implementado con `docker compose`, como se ha explicado anteriormente.

Puede probar con el explorador si usa CURL desde el terminal, como se muestra en la figura 4-30.

```
> curl http://localhost:51080/WeatherForecast


StatusCode        : 200
StatusDescription : OK
Content           : [{"date":"2021-01-06T12:13:43.7721387+00:00","temperatureC":42,"temperatureF":107,"summary":"Chilly"},{"date":"2021-01-07T12:1
                    ry":"Bracing ...
RawContent        : HTTP/1.1 200 OK
                    Transfer-Encoding: chunked
                    Content-Type: application/json; charset=utf-8
                    Date: Tue, 05 Jan 2021 12:13:43 GMT
                    Server: Kestrel

                    [{"date":"2021-01-06T12:13:43.7721387+00:00","temper ...
Forms             : {}
Headers           : {[Transfer-Encoding, chunked], [Content-Type, application/json; charset=utf-8], [Date, Tue, 05 Jan 2021 12:13:43 GMT], [Server
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 512
```

Figura 4-30. Prueba de una aplicación de Docker en local mediante CURL

Depuración de un contenedor que se ejecuta en Docker

Visual Studio Code admite la depuración de Docker si usa Node.js y otras plataformas como, por ejemplo, contenedores de .NET.

También puede depurar contenedores de .NET o .NET Framework en Docker cuando se usa Visual Studio para

Windows o Mac, tal y como se describe en la sección siguiente.

TIP

Para más información sobre la depuración de contenedores de Docker de Node.js, consulte https://blog.docker.com/2016/07/live-debugging-docker/ y /archive/blogs/user_ed/visual-studio-code-new-features-13-big-debugging-updates-rich-object-hover-conditional-breakpoints-node-js-mono-more.

Uso de herramientas de Docker de Visual Studio en Windows

06/03/2021 • 9 minutes to read • Edit Online

El flujo de trabajo del desarrollador al utilizar las herramientas de Docker incluidas en Visual Studio 2017, versión 15.7 y posteriores, es similar al de usar Visual Studio Code y la CLI de Docker (de hecho, se basa en la misma CLI de Docker), pero es más fácil de iniciar, simplifica el proceso y aumenta la productividad en las tareas de compilación, ejecución y composición. También puede ejecutar y depurar los contenedores a través de las teclas `F5` y `Ctrl+F5` habituales de Visual Studio. Incluso puede depurar una solución completa si sus contenedores se definen en el mismo archivo `docker-compose.yml` en el nivel de solución.

Configuración del entorno local

Con las últimas versiones de Docker para Windows, resulta más fácil que nunca desarrollar aplicaciones de Docker porque la configuración es sencilla, tal y como se explica en las siguientes referencias.

> **TIP**
> Para más información sobre cómo instalar Docker para Windows, vaya a (https://docs.docker.com/docker-for-windows/).

Compatibilidad con Docker en Visual Studio

Hay dos niveles de compatibilidad con Docker que puede agregar a un proyecto. En los proyectos de ASP.NET Core, basta con agregar un archivo `Dockerfile` al proyecto habilitando la compatibilidad con Docker. El siguiente nivel es la compatibilidad con la orquestación de contenedores, que agrega un archivo `Dockerfile` al proyecto (si aún no existe) y un archivo `docker-compose.yml` en el nivel de solución. La compatibilidad con la orquestación de contenedores, mediante Docker Compose, se agrega de forma predeterminada en las versiones 15.0 a 15.7 de Visual Studio 2017. La compatibilidad con la orquestación de contenedores es una característica opcional en Visual Studio 2017, versión 15.8 o posteriores. Visual Studio 2019 y versiones posteriores también admiten la implementación de **Kubernetes/Helm**.

Los comandos **Agregar > Compatibilidad con Docker** y **Agregar > Compatibilidad con el orquestador de contenedores** se encuentran en el menú contextual del nodo de proyecto de un proyecto de ASP.NET Core en el **Explorador de soluciones**, tal y como se muestra en la figura 4-31:

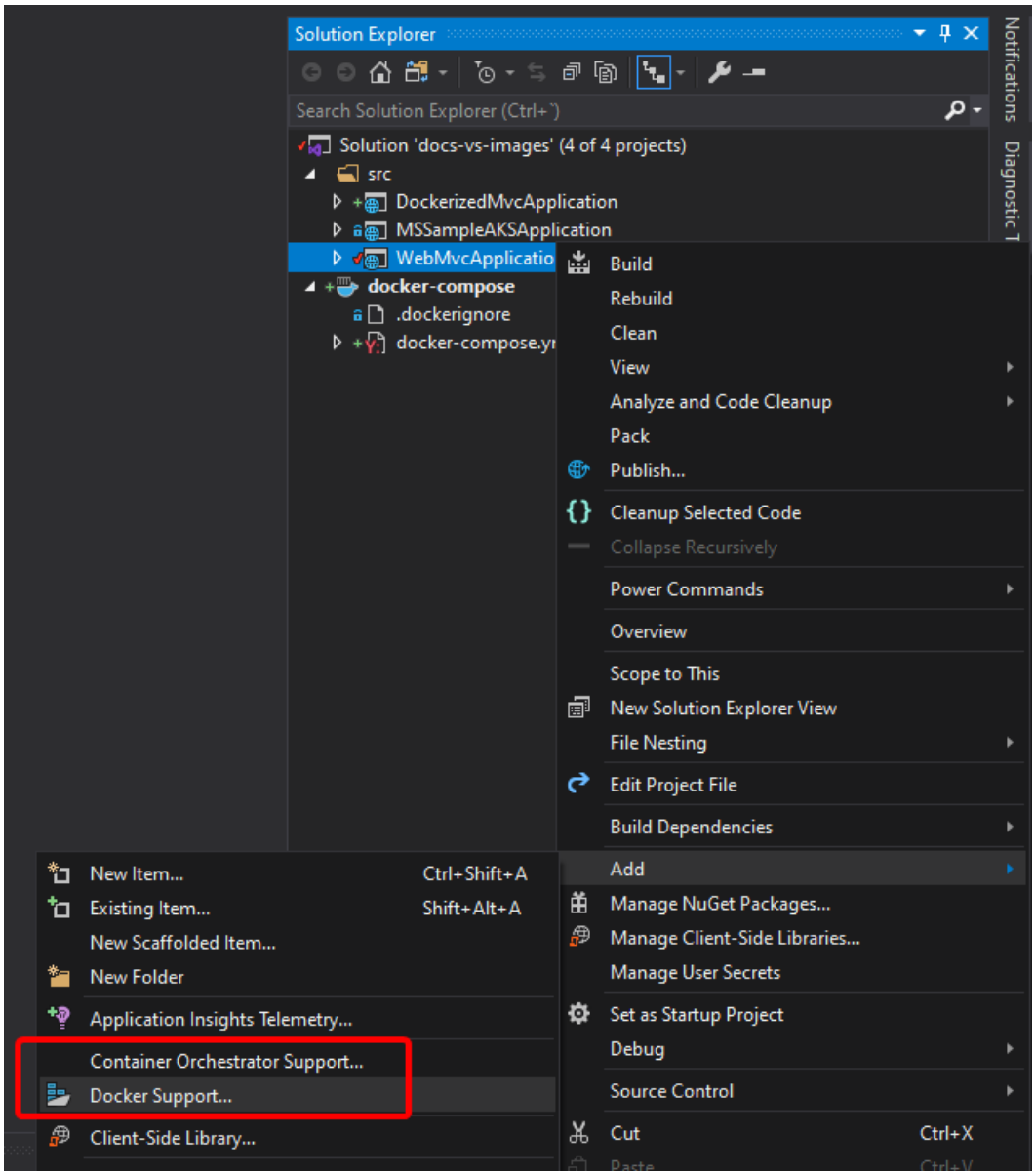


Figura 4-31. Incorporación de compatibilidad con Docker a un proyecto de Visual Studio 2019

Agregue compatibilidad con Docker

Además de la opción para agregar compatibilidad con Docker a una aplicación existente, como se ha mostrado en la sección anterior, también puede habilitar la compatibilidad con Docker durante la creación del proyecto si selecciona **Habilitar compatibilidad con Docker** en el cuadro de diálogo **Nueva aplicación web ASP.NET Core** que se abre al hacer clic en **Aceptar** en el cuadro de diálogo **Nuevo proyecto**, como se muestra en la figura 4-32.

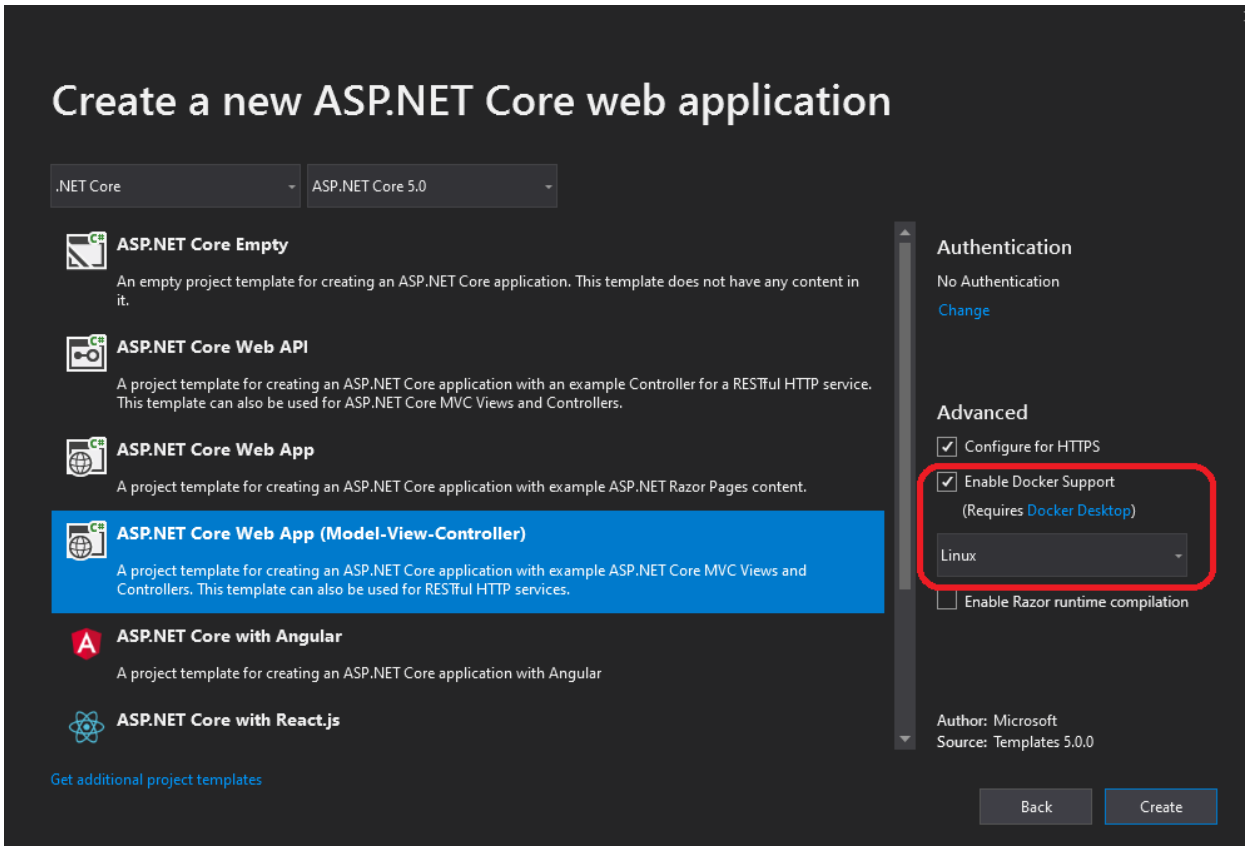


Figura 4-32. Habilitación de la compatibilidad con Docker durante la creación del proyecto en Visual Studio 2019

Al agregar o habilitar la compatibilidad con Docker, Visual Studio agrega al proyecto un archivo *Dockerfile* que incluye referencias a todos los proyectos necesarios de la solución.

Incorporación de compatibilidad con la orquestación de contenedores

Cuando quiera crear una solución de varios contenedores, agregue compatibilidad con la orquestación de contenedores a sus proyectos. De esta manera podrá ejecutar y depurar un grupo de contenedores (una solución completa) al mismo tiempo si se definen en el mismo archivo *docker-compose.yml*.

Para agregar compatibilidad con la orquestación de contenedores, haga clic con el botón derecho en el nodo de la solución o del proyecto en el **Explorador de soluciones** y elija **Agregar > Compatibilidad con la orquestación de contenedores**. Luego seleccione **Kubernetes/Helm** o **Docker Compose** para administrar los contenedores.

Después de agregar al proyecto compatibilidad con la orquestación de contenedores, ve que se han agregado un archivo Dockerfile al proyecto y una carpeta **docker-compose** a la solución en el **Explorador de soluciones**, como se muestra en la figura 4-33:

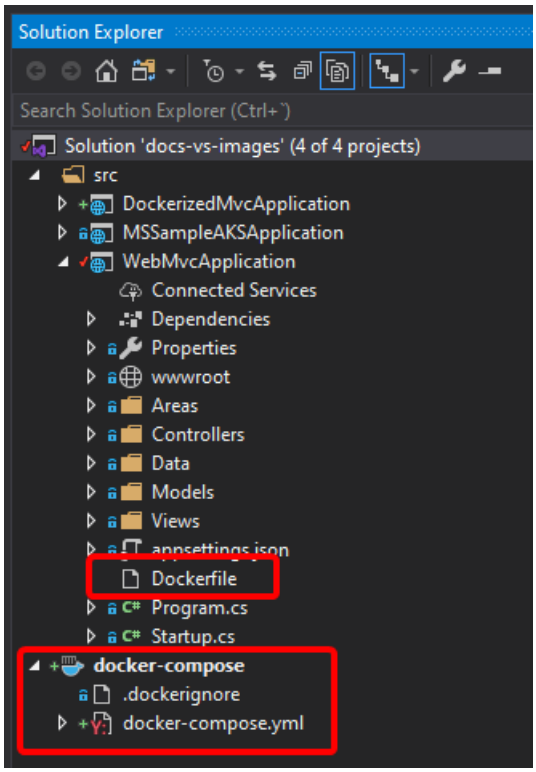


Figura 4-33. Archivos de Docker en el Explorador de soluciones de Visual Studio 2019

Si *docker-compose.yml* ya existe, Visual Studio le agrega simplemente las líneas de código de configuración necesarias.

Configuración de herramientas de Docker

En el menú principal, elija **Herramientas** > **Opciones** y expanda **Herramientas de contenedor** > **Configuración**. Aparecerá la configuración de las herramientas de contenedor.

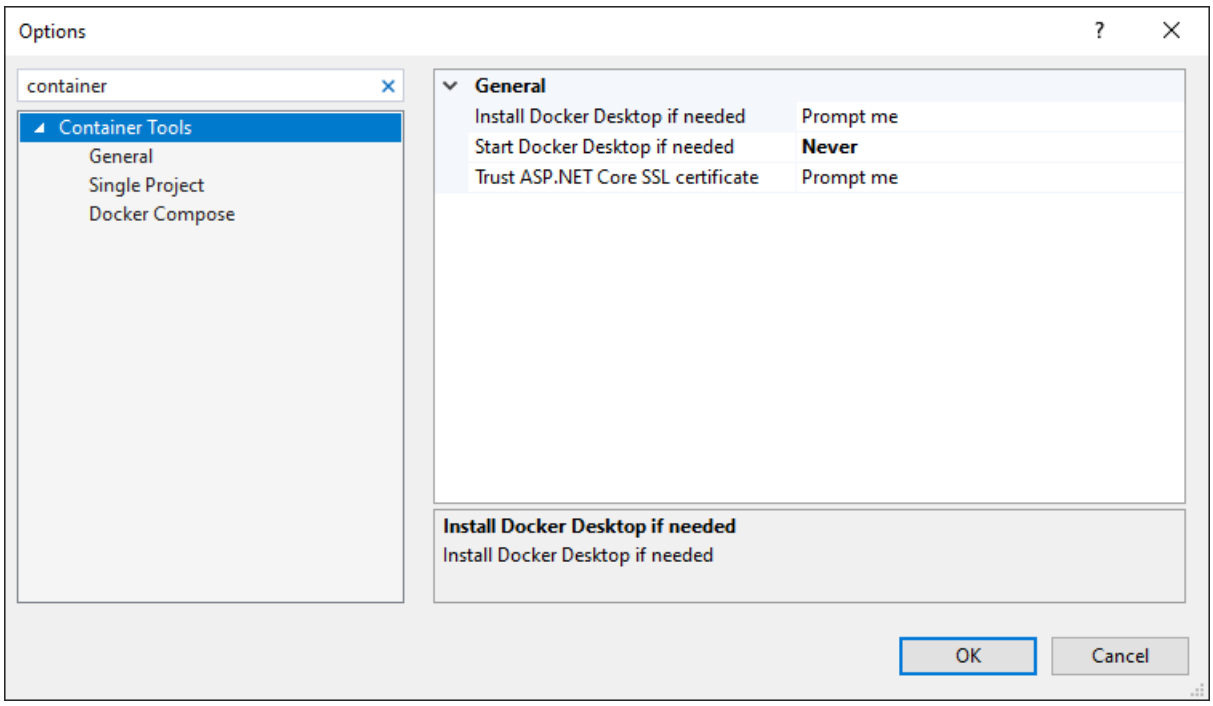


Figura 4-34. Opciones de herramientas de Docker

La tabla siguiente puede ayudarle a decidir cómo configurar estas opciones.

PÁGINA O VALOR	VALOR PREDETERMINADO	DESCRIPCIÓN
Página General		
Instalar Docker Desktop si es necesario	Preguntarme	
Iniciar Docker Desktop si es necesario	Preguntarme	
Confiar en el certificado SSL de ASP.NET Core	Preguntarme	Si el certificado SSL de localhost no se ha marcado como de confianza (con `dotnet dev-certs https --trust`), Visual Studio le pregunta cada vez que ejecuta el proyecto.
Página Proyecto único		
Extraer las imágenes de Docker necesarias al abrir el proyecto	True	Para un mejor rendimiento al ejecutar el proyecto, Visual Studio inicia una operación pull de Docker en segundo plano para que cuando esté listo para ejecutar el código, la imagen ya esté descargada o en proceso de descarga. Si simplemente carga proyectos y explora código, puede desactivar esta opción para evitar la descarga de imágenes de contenedor que no son necesarias. Esto puede ralentizar la experiencia del usuario de apertura del proyecto.
Extraer las imágenes de Docker actualizadas al abrir el proyecto	Proyectos de .NET Core	Extraiga las actualizaciones de las imágenes existentes para obtener las últimas actualizaciones al abrir el proyecto. Esto puede ralentizar la experiencia del usuario de apertura del proyecto.
Quitar los contenedores al cerrar el proyecto	True	Limpie al cerrar el proyecto. Esto puede ralentizar la experiencia del usuario de cierre del proyecto, pero suele ser rápido de todos modos.
Ejecutar contenedores al abrir el proyecto	True	Para un mejor rendimiento al ejecutar el proyecto, Visual Studio inicia todos los contenedores de la solución. Esto puede ralentizar la experiencia del usuario de apertura del proyecto.
Docker Compose		La página Docker Compose contiene la misma configuración que la página Proyecto único, pero se aplica a las soluciones de varios contenedores.

WARNING

Si el certificado SSL de localhost no es de confianza y la opción se establece en **Nunca**, se puede producir un error de las solicitudes web HTTPS en tiempo de ejecución en la aplicación o el servicio. En ese caso, vuelva a establecer el valor en **Preguntarme** o, mejor aún, confíe en los certificados del equipo de desarrollo mediante el comando `dotnet dev-certs https --trust`.

TIP

Para más información sobre la implementación de servicios y el uso de Visual Studio Tools para Docker, lea los artículos siguientes:

Depuración de aplicaciones en un contenedor de Docker local: /visualstudio/containers/edit-and-refresh

Implementación de un contenedor ASP.NET en un registro de contenedor con Visual Studio: /visualstudio/containers/hosting-web-apps-in-docker

ANTERIOR SIGUIENTE

Uso de comandos de Windows PowerShell en un archivo DockerFile para configurar contenedores con Windows (basado en Docker estándar)

06/03/2021 • 2 minutes to read • Edit Online

Con los contenedores de Windows, puede convertir las aplicaciones de Windows existentes en imágenes de Docker e implementarlas con las mismas herramientas que el resto del ecosistema de Docker.

Para usar contenedores de Windows, simplemente debe escribir comandos de Windows PowerShell en el archivo Dockerfile, como se muestra en el ejemplo siguiente:

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019
LABEL Description="IIS" Vendor="Microsoft" Version="10"
RUN powershell -Command Add-WindowsFeature Web-Server
CMD [ "ping", "localhost", "-t" ]
```

En este caso, vamos a usar Windows PowerShell para instalar una imagen base de Windows Server Core, así como IIS.

Del mismo modo, también se pueden usar comandos de Windows PowerShell para configurar componentes adicionales como ASP.NET 4.x tradicional, .NET Framework 4.6 o cualquier otro software de Windows, como se muestra aquí:

```
RUN powershell add-windowsfeature web-asp-net45
```

ANTERIOR SIGUIENTE

Compilación de aplicaciones ASP.NET Core implementadas como contenedores de Linux en un orquestador de AKS/Kubernetes

06/03/2021 • 15 minutes to read • Edit Online

Azure Kubernetes Service (AKS) es un servicio de orquestaciones de Kubernetes administrado de Azure que simplifica la implementación y la administración de contenedores.

Estas son las características principales de AKS:

- Plano de control hospedado en Azure
- Actualizaciones automatizadas
- Recuperación automática
- Escalado configurable por el usuario
- Experiencia de usuario más sencilla para desarrolladores y operadores de clúster.

En los ejemplos siguientes se examina la creación de una aplicación de ASP.NET Core 5.0 que se ejecuta en Linux y se implementa en un clúster de AKS en Azure, mientras que el desarrollo se realiza con Visual Studio 2019, versión 16.8.

Creación del proyecto de ASP.NET Core con Visual Studio 2019

ASP.NET Core es una plataforma de desarrollo de uso general de cuyo mantenimiento se encargan Microsoft y la comunidad .NET en GitHub. Es multiplataforma, admite Windows, macOS y Linux y puede usarse en escenarios de dispositivo, nube, IoT e incrustados.

En este ejemplo se usan un par de proyectos simples basados en plantillas de Visual Studio, por lo que no necesita muchos conocimientos adicionales para crear el ejemplo. Solo tiene que crear el proyecto con una plantilla estándar que incluya todos los elementos para ejecutar un proyecto pequeño con una API REST y una aplicación web con Razor Pages, mediante la tecnología ASP.NET Core 5.0.

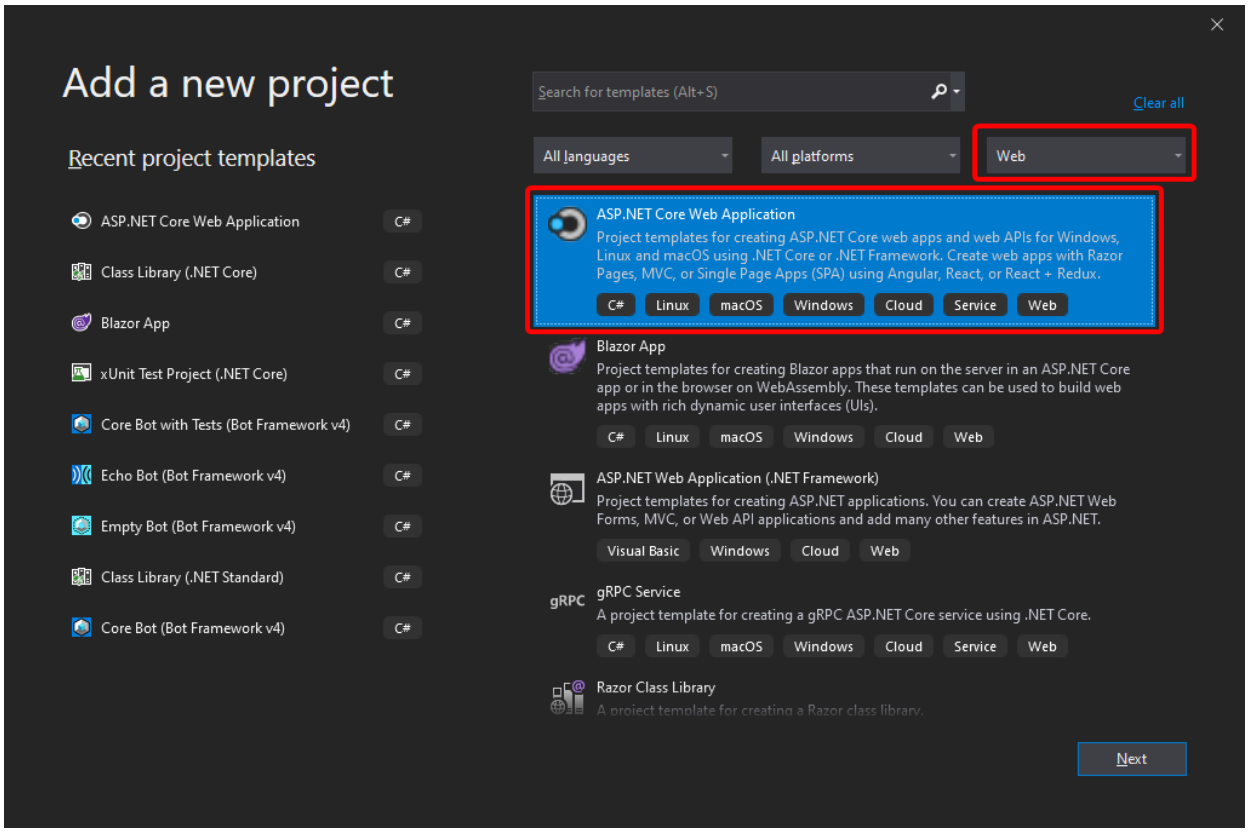


Figura 4-35. Creación de una aplicación web ASP.NET Core en Visual Studio 2019.

Para crear el proyecto de ejemplo en Visual Studio, seleccione **Archivo** > **Nuevo** > **Proyecto**, seleccione el tipo de proyecto **Web** y luego la plantilla **Aplicación web ASP.NET Core**. También puede buscar la plantilla si la necesita.

Luego escriba el nombre y la ubicación de la aplicación como se muestra en la siguiente imagen.

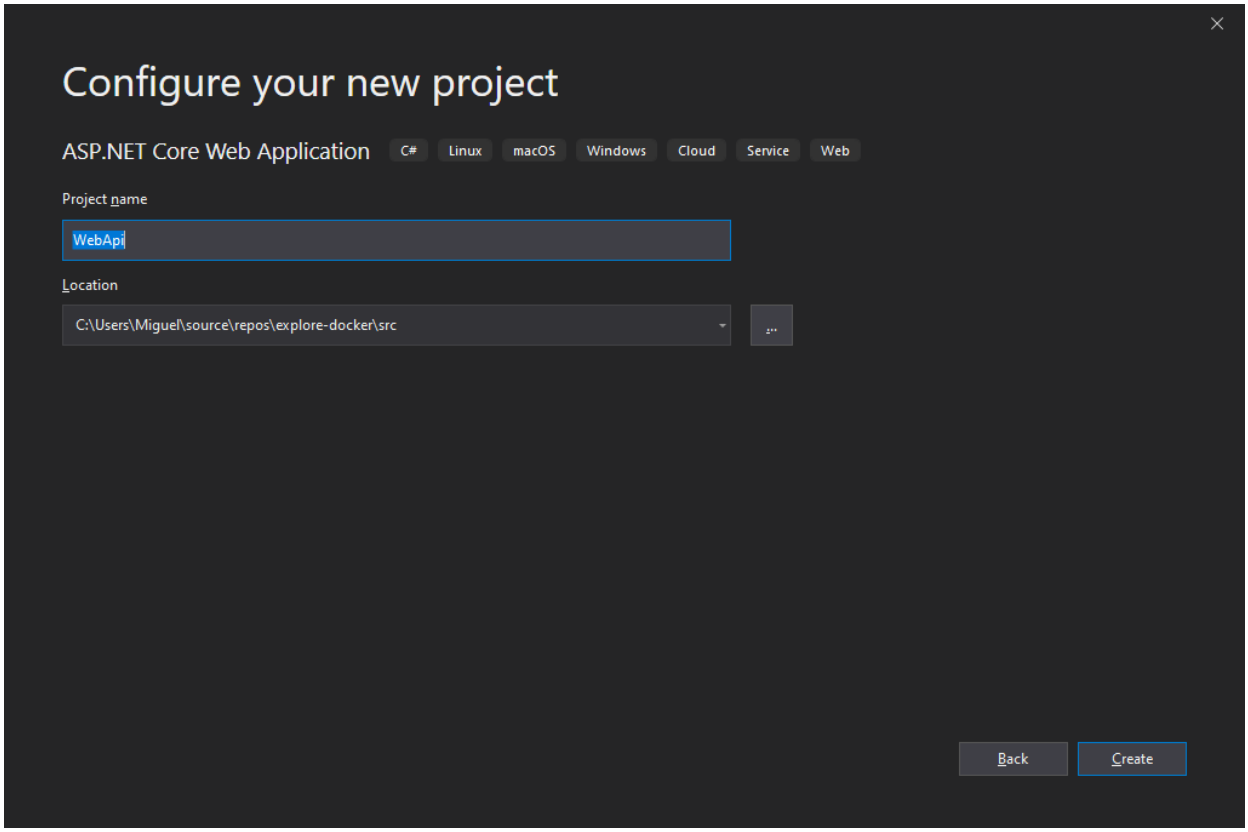


Figura 4-36. Especificación del nombre y la ubicación del proyecto en Visual Studio 2019.

Compruebe que ha seleccionado ASP.NET Core 5.0 como marco. .NET 5.0 está incluido en la última versión de

Visual Studio 2019, y se instala y configura automáticamente al instalar Visual Studio.

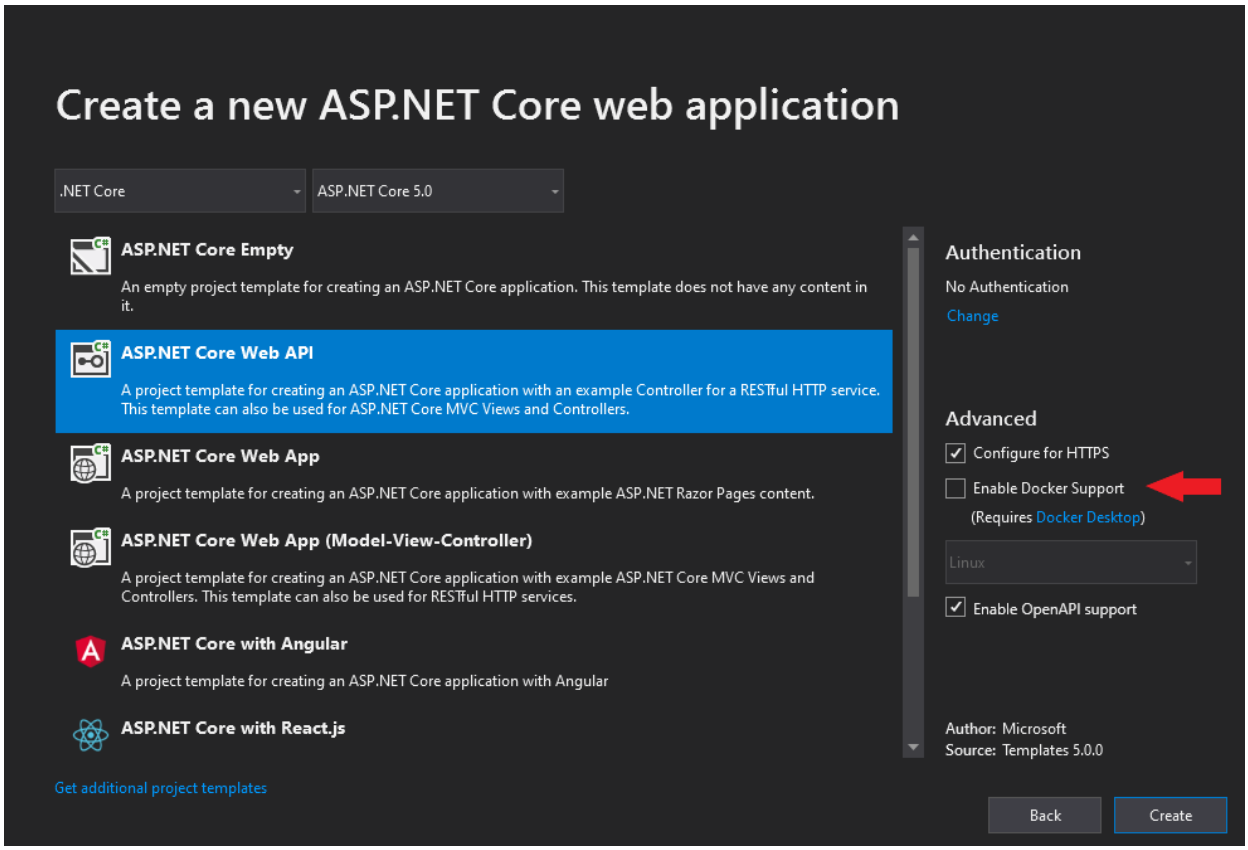


Figura 4-37. Selección de ASP.NET Core 5.0 y del tipo de proyecto API web

Observe que la compatibilidad con Docker no está habilitada, solo para mostrar que se puede hacer después de la creación del proyecto.

Si tiene alguna versión anterior de .NET Core, puede descargar e instalar la versión 3.1 desde https://dotnet.microsoft.com/download.

Para mostrar que puede "aplicar Docker" al proyecto en cualquier momento, se va a agregar compatibilidad con Docker ahora. Haga clic con el botón derecho en el nodo del proyecto en el Explorador de soluciones y seleccione **Agregar** > **Compatibilidad con Docker** en el menú contextual.

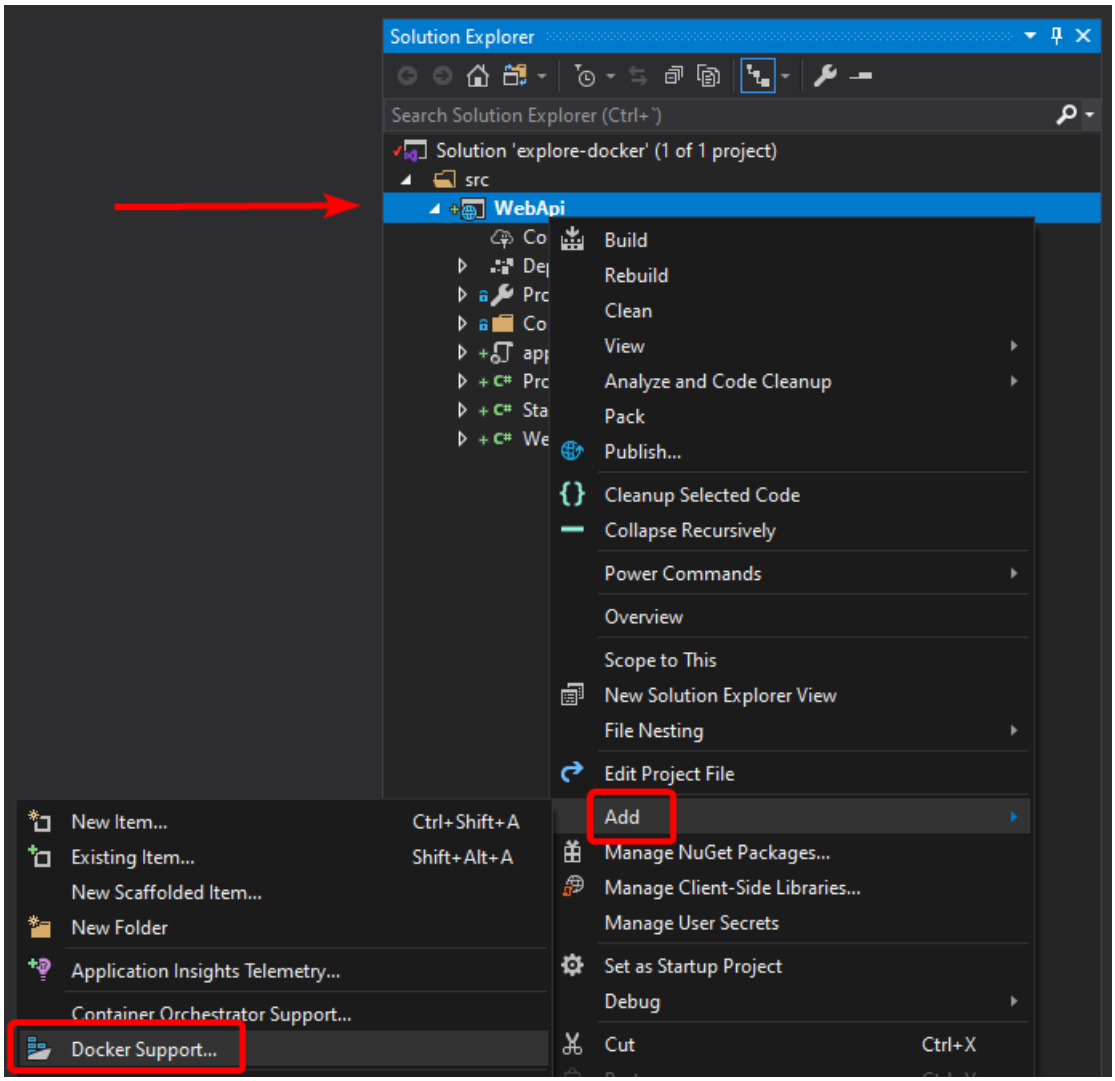


Figura 4-38. Incorporación de compatibilidad con Docker a un proyecto existente

Para acabar de agregar compatibilidad con Docker, puede elegir Windows o Linux. En este caso, seleccione **Linux**.

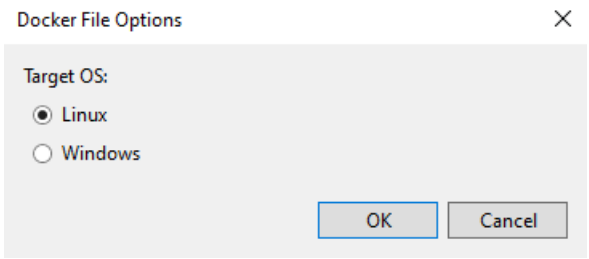


Figura 4-39. Selección de contenedores de Linux

Con estos sencillos pasos, ya tiene la aplicación de ASP.NET Core 5.0 en ejecución en un contenedor de Linux.

De forma similar, también puede agregar un proyecto **WebApp** muy sencillo (figura 4-40) para usar el punto de conexión de la API web, aunque los detalles no se tratan aquí.

Después, agregue compatibilidad con el orquestador al proyecto **WebApi** como se muestra a continuación, en la imagen 4-40.

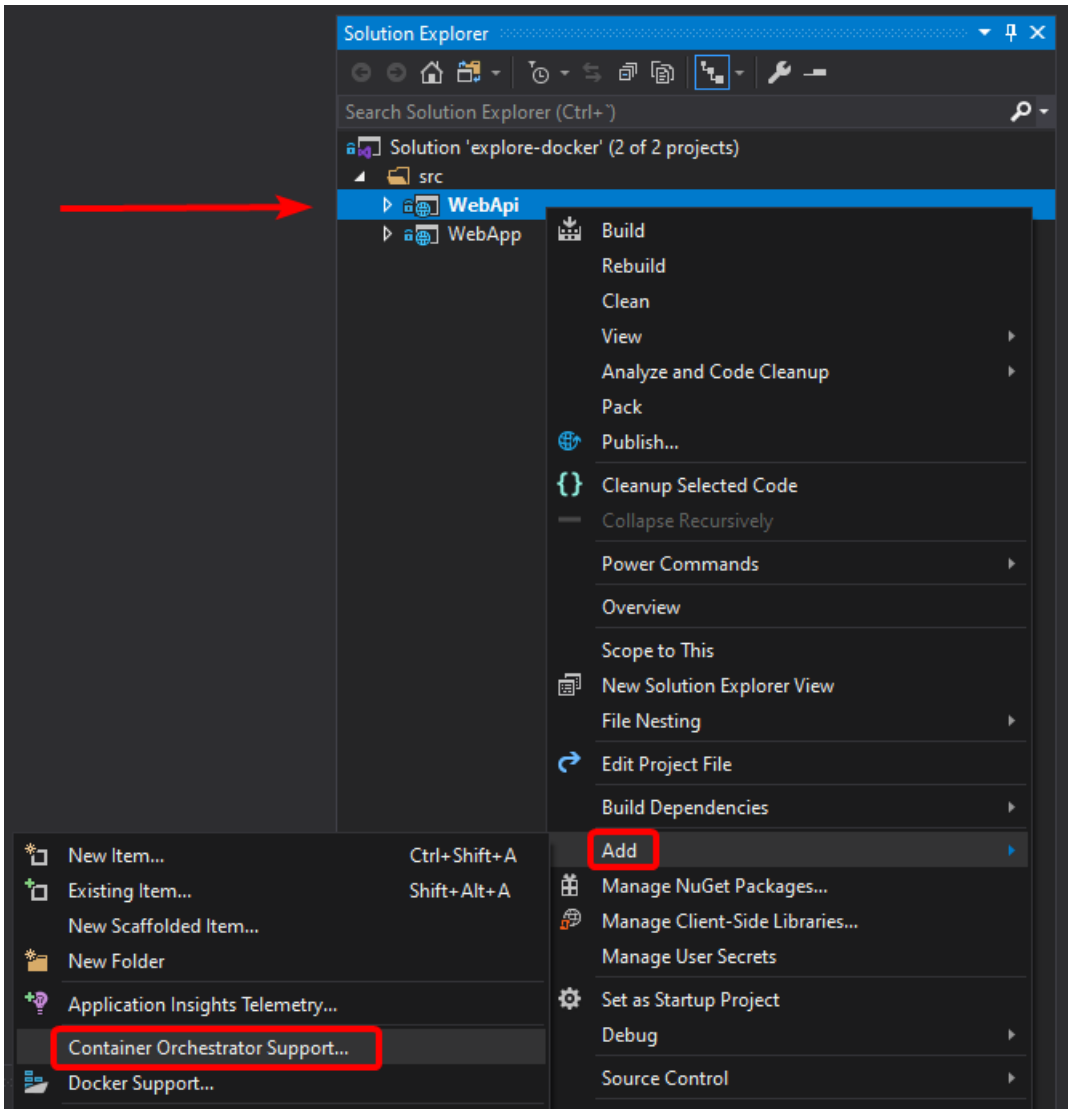


Figura 4-40. Incorporación de compatibilidad con el orquestador al proyecto *WebApi*.

Al seleccionar la opción `Docker Compose`, que es correcta para el desarrollo local, Visual Studio agrega el proyecto docker-compose, con los archivos de docker-compose, como se muestra en la imagen 4-41.

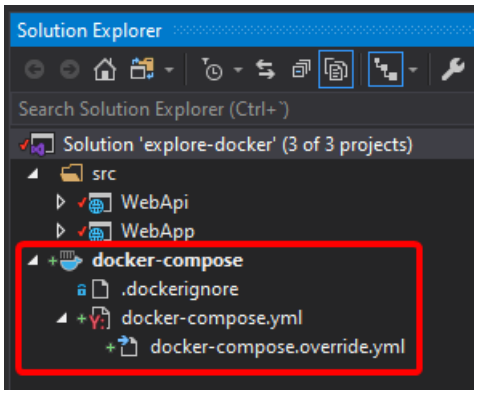


Figura 4-41. Incorporación de compatibilidad con el orquestador al proyecto *WebApi*.

Los archivos iniciales agregados son similares a estos:

`docker-compose.yml`

```
version: "3.4"

services:
  webapi:
    image: ${DOCKER_REGISTRY-}webapi
    build:
      context: .
      dockerfile: WebApi/Dockerfile

  webapp:
    image: ${DOCKER_REGISTRY-}webapp
    build:
      context: .
      dockerfile: WebApp/Dockerfile
```

`docker-compose.override.yml`

```
version: "3.4"

services:
  webapi:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
    ports:
      - "80"
      - "443"
    volumes:
      - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
  webapp:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
    ports:
      - "80"
      - "443"
    volumes:
      - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
```

Para que la aplicación se ejecute con Docker Compose, solo tiene que realizar algunos ajustes en `docker-compose.override.yml`

```
services:
  webapi:
    #...
    ports:
      - "51080:80"
      - "51443:443"
    #...
  webapp:
    environment:
      #...
      - WebApiBaseAddress=http://webapi
    ports:
      - "50080:80"
      - "50443:443"
    #...
```

Ahora puede ejecutar la aplicación con la tecla **F5**, mediante el botón **Reproducir**, o bien la tecla **Ctrl+F5** al seleccionar el proyecto docker-compose, como se muestra en la imagen 4-42.

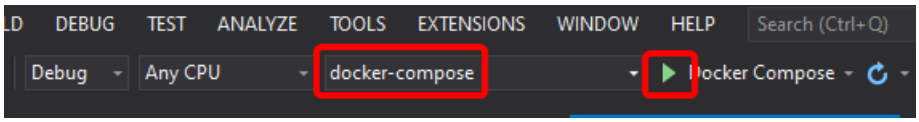


Figura 4-42. Incorporación de compatibilidad con el orquestador al proyecto *WebApi*.

Al ejecutar la aplicación docker-compose como se ha explicado, se consigue:

1. Compilar las imágenes y crear los contenedores según el archivo docker-compose.
2. Abrir el explorador en la dirección configurada en el cuadro de diálogo "Propiedades" del proyecto `docker-compose`.
3. Abrir la ventana **Contenedor** (en Visual Studio 2019, versión 16.4 y posteriores).
4. Compatibilidad del depurador con todos los proyectos de la solución, como se muestra en las siguientes imágenes.

Explorador abierto:

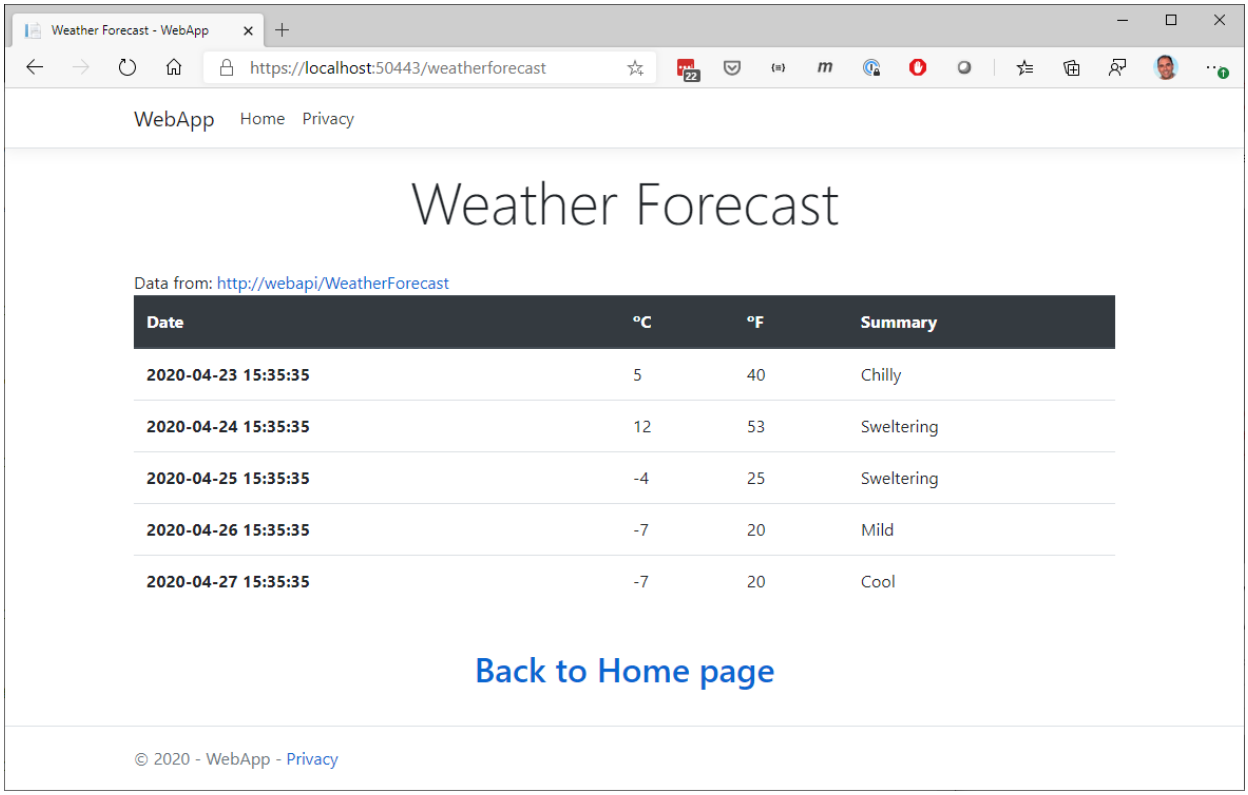


Figura 4-43. Ventana del explorador con una aplicación en ejecución en varios contenedores.

Ventana Contenedores:

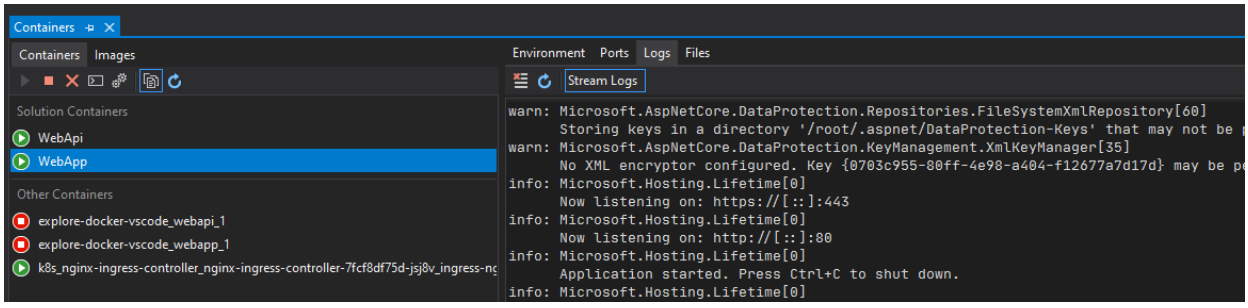


Figura 4-44. Ventana "Contenedores" de Visual Studio

La ventana **Contenedores** permite ver los contenedores en ejecución, examinar las imágenes disponibles, ver variables de entorno, registros y asignaciones de puertos, inspeccionar el sistema de archivos, adjuntar un depurador o abrir una ventana de terminal dentro del entorno del contenedor.

Como puede ver, la integración entre Visual Studio 2019 y Docker está totalmente orientada a la productividad del desarrollador.

Por supuesto, también puede enumerar las imágenes mediante el comando `docker images`. Debería ver las imágenes `webapi` y `webapp` con las etiquetas `dev` creadas por la implementación automática del proyecto con Visual Studio 2019.

```
docker images
```

```
docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
webapi                              latest              301ede061a1e        5 minutes ago       210MB
webapp                              latest              413e9355ed78        5 minutes ago       210MB
mcr.microsoft.com/dotnet/sdk        5.0                 acc2e9a7698d        25 hours ago        616MB
mcr.microsoft.com/dotnet/aspnet     5.0                 66792fe28528        25 hours ago        205MB
```

Figura 4-45. Vista de imágenes de Docker

Registro de la solución en una instancia de Azure Container Registry (ACR)

Puede cargar las imágenes en Azure Container Registry (ACR), pero también puede usar Docker Hub o cualquier otro registro de modo que las imágenes puedan implementarse en el clúster de AKS desde ese registro.

Creación de una instancia de ACR

Ejecute el siguiente comando desde **az cli**:

```
az acr create --name exploredocker --resource-group explore-docker-aks-rg --sku basic --admin-enabled
```

> **NOTE**
> El nombre del registro de contenedor (p. ej. `exploredocker`) debe ser único dentro de Azure y contener entre 5 y 50 caracteres alfanuméricos. Para obtener más información, vea Creación de un registro de contenedor

Creación de la imagen en modo de versión

Ahora va a crear la imagen en modo **Versión** (listo para producción). Para ello, cambie a **Versión**, como se muestra en la figura 4-46, y ejecute la aplicación como antes.

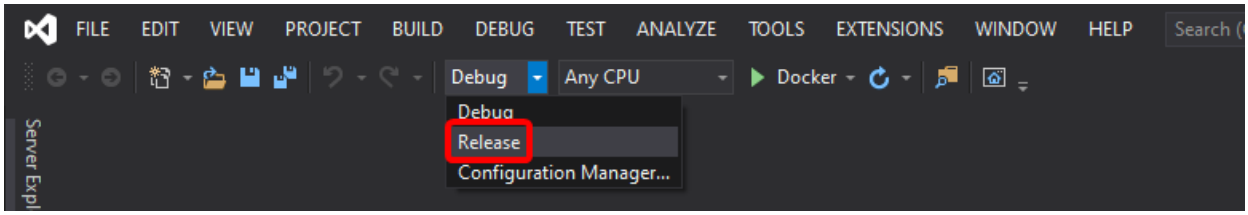


Figura 4-46. Selección del modo de versión

Si ejecuta el comando `docker images`, ve que se crean dos imágenes, una para el modo `debug` (**desarrollo**) y otra para el modo `release` (**más reciente**).

Creación de una nueva etiqueta para la imagen

Es preciso etiquetar cada imagen de contenedor con el nombre `loginServer` del registro. Esta etiqueta se usa para el enrutamiento al insertar imágenes de contenedor en un registro de imágenes.

Para ver el nombre `loginServer` desde Azure Portal, tome la información de Azure Container Registry.

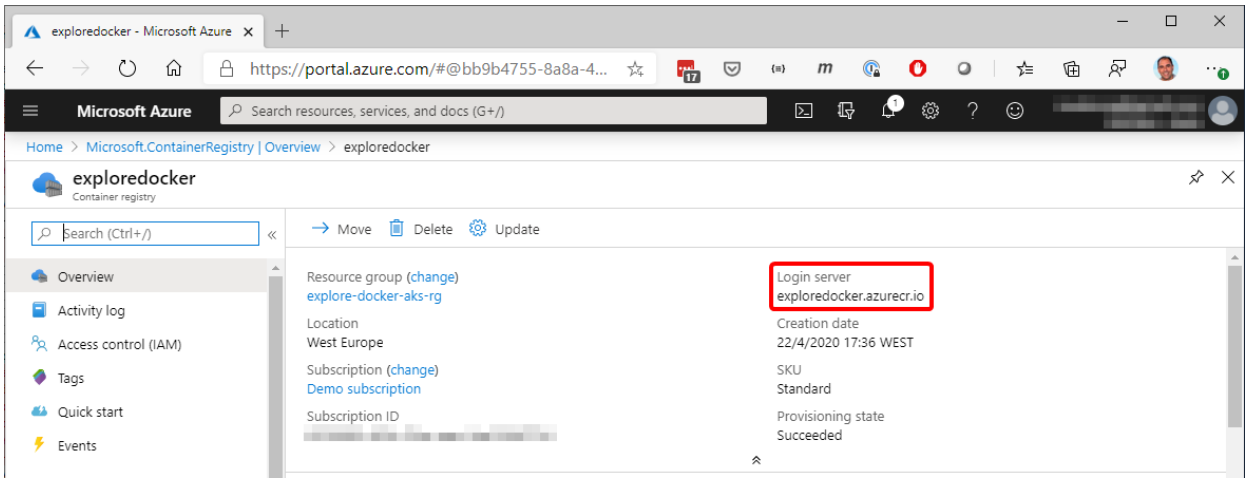


Figura 4-47. Vista del nombre del Registro

También puede ejecutar el comando siguiente:

```
az acr list --resource-group <resource-group-name> --query "[].{acrLoginServer:loginServer}" --output table
```

```
PS: [Miguel]>az acr list --resource-group explore-docker-aks-rg --query "[].{acrLoginServer:loginServer}" --output table
AcrLoginServer
-----------------------
exploredocker.azurecr.io
```

Figura 4-48. Obtención del nombre del registro mediante **az cli**

En ambos casos, obtendrá el nombre. En nuestro ejemplo, se trata de `exploredocker.azurecr.io`.

Ahora puede etiquetar la imagen. Para ello, tome la imagen más reciente (la imagen de versión) con este comando:

```
docker tag <image-name>:latest <login-server-name>/<image-name>:v1
```

Después de ejecutar el comando `docker tag`, muestre las imágenes con el comando `docker images`. Debería ver la imagen con la nueva etiqueta.

```
PS: [Miguel]>docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
webapp                                    latest              11f2a7d7718c        4 minutes ago       213MB
exploredocker.azurecr.io/webapp           v1                  11f2a7d7718c        4 minutes ago       213MB
webapi                                    latest              db9979d4cfef        4 minutes ago       208MB
exploredocker.azurecr.io/webapi           v1                  db9979d4cfef        4 minutes ago       208MB
webapp                                    dev                 cba194e20cb9        About an hour ago   207MB
webapi                                    dev                 ed6ea5466cb4        About an hour ago   207MB
```

Figura 4-49. Vista de las imágenes etiquetadas

Inserción de la imagen en Azure ACR

Inicie sesión en Azure Container Registry.

```
az acr login --name exploredocker
```

Inserte la imagen en Azure ACR con el comando siguiente:

```
docker push <login-server-name>/<image-name>:v1
```

Este comando tarda un poco en cargar las imágenes, pero proporciona información durante el proceso. En la imagen siguiente puede ver la salida de una imagen completada y otra en curso.

```
PS: [Miguel]>docker push exploredocker.azurecr.io/webapi:v1
The push refers to repository [exploredocker.azurecr.io/webapi]
35e6bc6eeb8d: Pushed
1754f2ec3a1a: Pushed
8957624a4d9d: Pushed
b21b0b2df887: Pushed
77362d5ab4dc: Pushed
974ff534c026: Pushed
b60e5c3bcef2: Pushed
v1: digest: sha256:0f9415358734702a89084b8461eed3c50047c5c347093c311eb8d32d15ad0fda size: 1791
PS: [Miguel]>docker push exploredocker.azurecr.io/webapp:v1
The push refers to repository [exploredocker.azurecr.io/webapp]
33e7f8019853: Pushing [================================================>]  5.322MB
1754f2ec3a1a: Preparing
8957624a4d9d: Preparing
b21b0b2df887: Preparing
77362d5ab4dc: Preparing
974ff534c026: Waitingng
b60e5c3bcef2: Waiting
```

Figura 4-50. Salida de la consola del comando push.

Para implementar la aplicación de varios contenedores en el clúster de AKS, necesita algunos archivos `.yaml` de manifiesto que tengan la mayoría de las propiedades tomadas de los archivos `docker-compose.yml` y `docker-compose.override.yml`.

`deploy-webapi.yml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapi
  labels:
    app: weather-forecast
spec:
  replicas: 1
  selector:
    matchLabels:
      service: webapi
  template:
    metadata:
      labels:
        app: weather-forecast
        service: webapi
    spec:
      containers:
        - name: webapi
          image: exploredocker.azurecr.io/webapi:v1
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
              protocol: TCP
          env:
            - name: ASPNETCORE_URLS
              value: http://+:80
---
apiVersion: v1
kind: Service
metadata:
  name: webapi
  labels:
    app: weather-forecast
    service: webapi
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    service: webapi
```

deploy-webapp.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  labels:
    app: weather-forecast
spec:
  replicas: 1
  selector:
    matchLabels:
      service: webapp
  template:
    metadata:
      labels:
        app: weather-forecast
        service: webapp
    spec:
      containers:
        - name: webapp
          image: exploredocker.azurecr.io/webapp:v1
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
              protocol: TCP
          env:
            - name: ASPNETCORE_URLS
              value: http://+:80
            - name: WebApiBaseAddress
              value: http://webapi
---
apiVersion: v1
kind: Service
metadata:
  name: webapp
  labels:
    app: weather-forecast
    service: webapp
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    service: webapp
```

NOTE

Los archivos `.yml` anteriores solo habilitan los puertos `HTTP` mediante el parámetro `ASPNETCORE_URLS` para evitar problemas con el certificado que falta en la aplicación de ejemplo.

TIP

Si quiere saber cómo se crea el clúster de AKS para este ejemplo, vea la sección **Implementación en Azure Kubernetes Service (AKS)** de esta guía.

Ya está casi listo para implementar con **kubectl**, pero primero debe obtener las credenciales del clúster de AKS con este comando:

```
az aks get-credentials --resource-group explore-docker-aks-rg --name explore-docker-aks
```

```
PS: [Miguel]>az aks get-credentials --resource-group explore-docker-aks-rg --name explore-docker-aks
Merged "explore-docker-aks" as current context in C:\Users\Miguel\.kube\config
```

Figura 4-51. Obtención de credenciales de AKS en el entorno de kubectl.

También debe permitir que el clúster de AKS extraiga imágenes de la instancia de ACR con este comando:

```
az aks update --name explore-docker-aks --resource-group explore-docker-aks-rg --attach-acr exploredocker
```

El comando anterior puede tardar un par de minutos en terminar. Luego use el comando `kubectl apply` para iniciar las implementaciones; entonces, `kubectl get all` indica los objetos del clúster.

```
kubectl apply -f deploy-webapi.yml
kubectl apply -f deploy-webapp.yml

kubectl get all
```

```
PS: [Miguel]>kubectl apply -f deploy-webapi.yml
deployment.apps/webapi created
service/webapi created
PS: [Miguel]>kubectl apply -f deploy-webapp.yml
deployment.apps/webapp created
service/webapp created
PS: [Miguel]>kubectl get all
NAME                          READY   STATUS              RESTARTS   AGE
pod/webapi-676467cfc8-wqg5s   0/1     ContainerCreating   0          15s
pod/webapp-56cfc7fdb5-cnxsd   0/1     ContainerCreating   0          7s


NAME                 TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP      10.0.0.1      <none>        443/TCP        4h33m
service/webapi       ClusterIP      10.0.97.189   <none>        80/TCP         15s
service/webapp       LoadBalancer   10.0.74.46    <pending>     80:31349/TCP   7s


NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/webapi   0/1     1            0           15s
deployment.apps/webapp   0/1     1            0           7s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/webapi-676467cfc8   1         1         0       15s
replicaset.apps/webapp-56cfc7fdb5   1         1         0       7s
```

Figura 4-52. Implementación en Kubernetes

Tiene que esperar un rato para que el equilibrador de carga obtenga la dirección IP externa al consultar con `kubectl get services` y, luego, la aplicación debe estar disponible en esa dirección, como se muestra en la siguiente imagen:

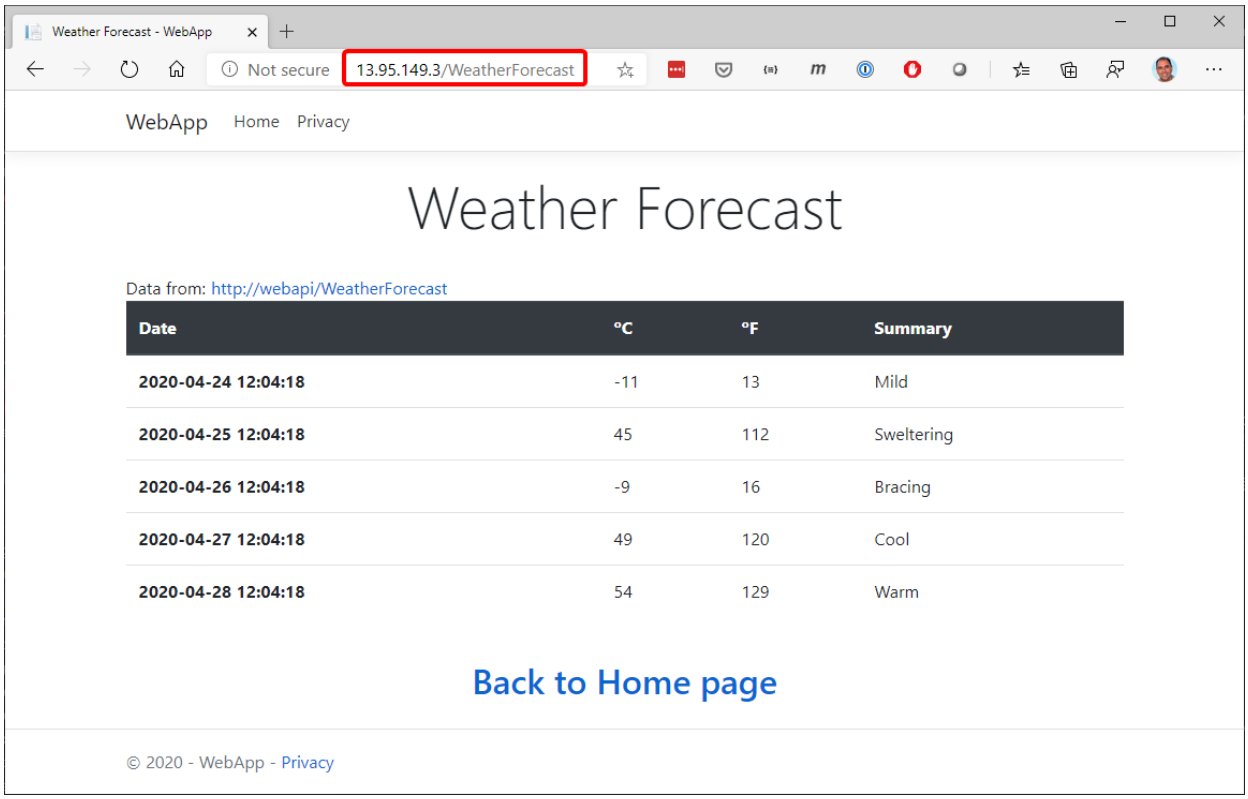


Figura 4-53. Implementación en Kubernetes

Una vez finalizada la implementación, puede acceder a la interfaz de usuario web de Kubernetes con un proxy local, mediante un túnel SSH.

En primer lugar, debe crear un elemento ClusterRoleBinding con el siguiente comando:

```
kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
```

Y luego este comando para iniciar el proxy:

```
az aks browse --resource-group exploredocker-aks-rg --name explore-docker-aks
```

Se debe abrir una ventana del explorador en `http://127.0.0.1:8001` con una vista similar a esta:

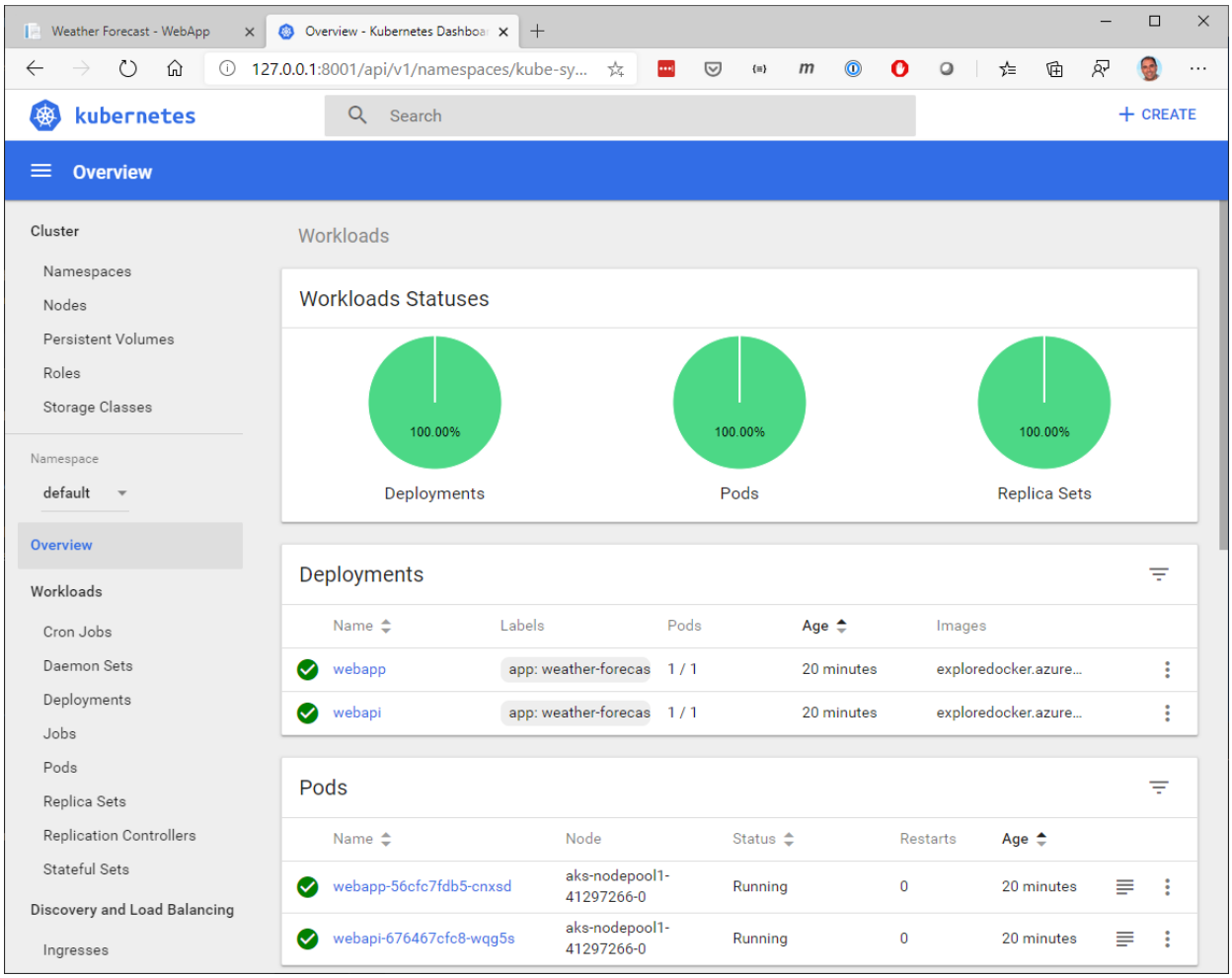


Figura 4-54. Vista de la información del clúster de Kubernetes

Ya tiene la aplicación ASP.NET Core en ejecución en contenedores de Linux e implementada en un clúster de AKS en Azure.

> **NOTE**
>
> Para obtener más información sobre la implementación con Kubernetes, vea https://kubernetes.io/docs/reference/kubectl/cheatsheet/.

Flujo de trabajo de DevOps para aplicaciones de Docker con herramientas de Microsoft

06/03/2021 • 4 minutes to read • Edit Online

Microsoft Visual Studio, Azure DevOps Services, Team Foundation Server y Azure Monitor proporcionan un ecosistema completo para el desarrollo y las operaciones de TI que proporcionan a su equipo las herramientas para administrar proyectos y compilar, probar e implementar rápidamente las aplicaciones en contenedor.

Con Visual Studio y Azure DevOps Services en la nube, junto con Team Foundation Server local, los equipos de desarrollo pueden crear, probar y publicar aplicaciones en contenedor de manera productiva dirigidas a Windows o Linux.

Las herramientas de Microsoft pueden automatizar la canalización de implementaciones específicas de aplicaciones contenedorizadas (Docker, .NET o cualquier combinación con otras plataformas), desde compilaciones globales e integración continua (CI) y pruebas con Azure DevOps Services o Team Foundation Server, para realizar una implementación continua (CD) en entornos de Docker (desarrollo, almacenamiento provisional y producción) y para transmitir información de análisis sobre los servicios al equipo de desarrollo por medio de Azure Monitor. Cada confirmación de código puede iniciar una compilación (CI) e implementar automáticamente los servicios en entornos en contenedores específicos (CD).

Los desarrolladores y evaluadores pueden aprovisionar con facilidad y rapidez entornos de prueba y desarrollo tipo producción basados en Docker con el uso de plantillas de Microsoft Azure.

La complejidad del desarrollo de aplicaciones en contenedores aumenta de forma constante según las necesidades empresariales de complejidad y escalabilidad. Un buen ejemplo de esta complejidad son las aplicaciones basadas en arquitecturas de microservicios. Para tener éxito en un entorno de este tipo, el proyecto debe automatizar todo el ciclo de vida, no solo la compilación y el desarrollo, sino que también debe administrar versiones junto con la colección de telemetría. Azure DevOps Services y Azure ofrecen las siguientes funcionalidades:

- Administración del código fuente de Azure DevOps Services/Team Foundation Server (basada en Git o en el Control de versiones de Team Foundation), planeación de Agile (Agile, Scrum y CMMI son compatibles), integración continua, administración de versiones y otras herramientas para equipos de Agile.
- Azure DevOps Services y Team Foundation Server incluyen un eficaz y creciente ecosistema de extensiones propias y de terceros con las que puede crear con facilidad una integración continua, así como compilar, probar, entregar y publicar la canalización de administración de microservicios.
- Ejecución de pruebas automatizadas como parte de la canalización de compilación de Azure DevOps Services.
- Azure DevOps Services puede reforzar el ciclo de vida de DevOps con la entrega a varios entornos, no solo para entornos de producción, sino también para pruebas, incluida la experimentación de A/B, versiones de valor controlado, etc.
- Las organizaciones pueden aprovisionar fácilmente contenedores de Docker en imágenes privadas almacenadas en Azure Container Registry junto con cualquier dependencia de componentes de Azure (Data, PaaS, etc.) mediante el uso de plantillas de Azure Resource Manager con herramientas con las que se sienten cómodas al trabajar.

ANTERIOR SIGUIENTE

Pasos del flujo de trabajo de DevOps de bucle externo para una aplicación de Docker

06/03/2021 • 31 minutes to read • Edit Online

En la figura 5-1 se muestra una representación de un extremo a otro de los pasos que componen el flujo de trabajo del bucle externo de DevOps. Muestra el "bucle externo" de DevOps. Cuando se inserta código en el repositorio, se inicia una canalización de CI y, después, comienza la canalización de CD, donde se implementa la aplicación. Las métricas recopiladas de las aplicaciones implementadas se envían a la carga de trabajo de desarrollo, donde se produce el "bucle interno", a fin de que los equipos de desarrollo tengan datos reales para responder a las necesidades del usuario y de la organización.

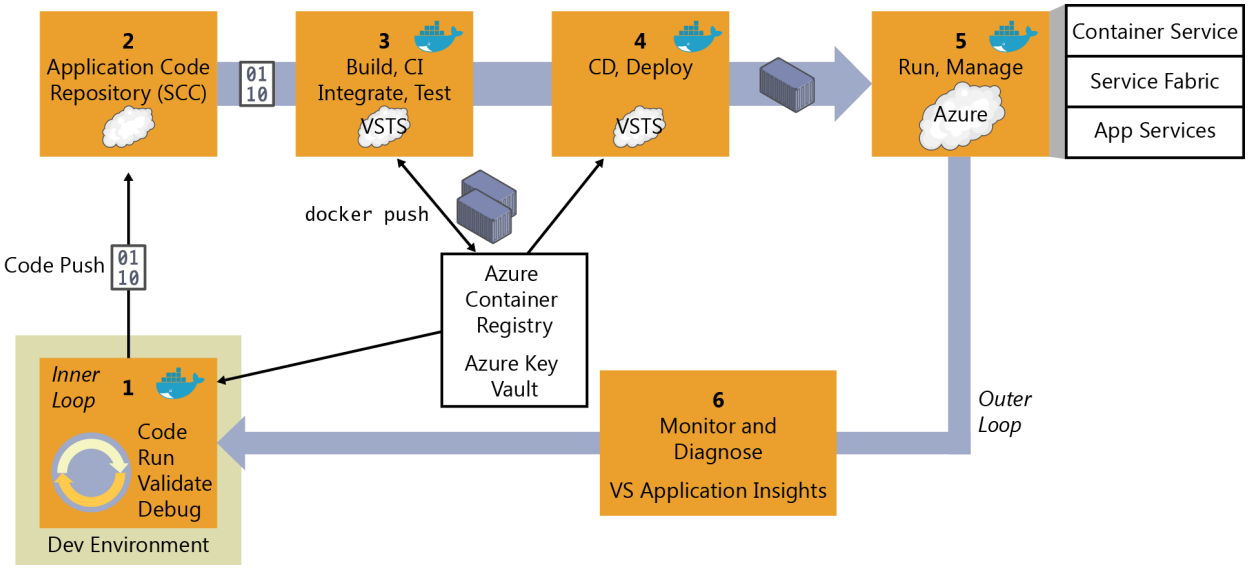


Figura 5-1. Flujo de trabajo del bucle externo de DevOps para aplicaciones de Docker con herramientas de Microsoft

Examinemos cada uno de estos pasos con mayor detalle.

Paso 1: flujo de trabajo de desarrollo del bucle interno

Este paso se explica con detalle en el capítulo 4, pero, a modo de resumen, aquí es donde empieza el bucle externo. En este momento, el desarrollador inserta código en el sistema de administración de control de código fuente (por ejemplo, GIT), lo que da inicio a acciones de la canalización de CI.

Paso 2: integración y administración del control de código fuente con Azure DevOps Services y GIT

En este paso, debe contar con un sistema de control de versiones para recopilar una versión consolidada de todo el código procedente de los diferentes desarrolladores del equipo.

Aunque el control de código fuente (SCC) y la administración de código fuente pueden parecerles algo instintivo a la mayoría de los desarrolladores, al crear aplicaciones de Docker en un ciclo de vida de DevOps, es fundamental hacer hincapié en que no se deben enviar las imágenes de Docker con la aplicación directamente al registro de Docker de global (como Azure Container Registry o Docker Hub) desde el equipo del desarrollador. Por el contrario, las imágenes de Docker que van a lanzarse e implementarse en entornos de producción deben crearse únicamente en el código fuente que se está integrando en la canalización de compilación o CI global, en

función del repositorio de código fuente (por ejemplo, GIT).

Las imágenes locales que generen los desarrolladores se usarán exclusivamente cuando estos prueben sus equipos. Por este motivo es fundamental que la canalización de DevOps esté activada desde el código de SCC.

Azure DevOps Services y Team Foundation Server admiten GIT y Control de versiones de Team Foundation. Puede elegir cualquiera de ellos y usarlo para disfrutar de una experiencia de Microsoft de un extremo a otro. Aun así, también puede administrar el código en repositorios externos (como GitHub, repositorios de GIT locales o Subversion) y seguir siendo capaz de conectarse a él y obtener el código como punto de partida para la canalización de CI de DevOps.

Paso 3: compilación, CI, integración y prueba con Azure DevOps Services y Docker

CI se ha convertido en un estándar para las pruebas y la entrega de software moderno. La solución de Docker mantiene una clara separación de intereses entre los equipos de desarrollo y operaciones. La inmutabilidad de las imágenes de Docker garantiza una implementación repetible entre lo que se ha desarrollado, lo que se ha probado a través de CI y lo que se ha ejecutado en producción. El motor de Docker implementado en los portátiles de los desarrolladores y la infraestructura de prueba hace que los contenedores puedan transportarse a distintos entornos.

Llegado a este punto, cuando tenga un sistema de control de versiones con el código correcto enviado, necesitará un *servicio de compilación* para tomar el código y ejecutar la compilación y las pruebas globales.

El flujo de trabajo interno para este paso (CI, compilación, prueba) consiste en la construcción de una canalización de CI que consta del repositorio de código (GIT, etc.), el servidor de compilación (Azure DevOps Services), el motor de Docker y un registro de Docker.

Puede usar Azure DevOps Services como base para compilar las aplicaciones y configurar la canalización de CI, así como para publicar los "artefactos" compilados en un "repositorio de artefactos", como se explica en el paso siguiente.

Al usar Docker para la implementación, los "artefactos finales" que se implementarán son imágenes de Docker en las que se ha insertado la aplicación o los servicios. Dichas imágenes se insertan o se publican en un *registro de Docker* (un repositorio privado, como los que se pueden tener en Azure Container Registry, o un repositorio público, como el registro de Docker Hub, que suele usarse para imágenes base oficiales).

Este es el concepto básico: la canalización de CI la iniciará una confirmación en un repositorio de SCC, como GIT. La confirmación hará que Azure DevOps Services ejecute un trabajo de compilación dentro de un contenedor de Docker y, tras la finalización correcta de ese trabajo, insertará una imagen de Docker en el registro de Docker, como se muestra en la figura 5-2. La primera parte del bucle externo implica los pasos del 1 a 3, es decir, primero el paso de programación, ejecución, depuración y validación, después el repositorio de código y, por último, la compilación, la CI y la prueba.

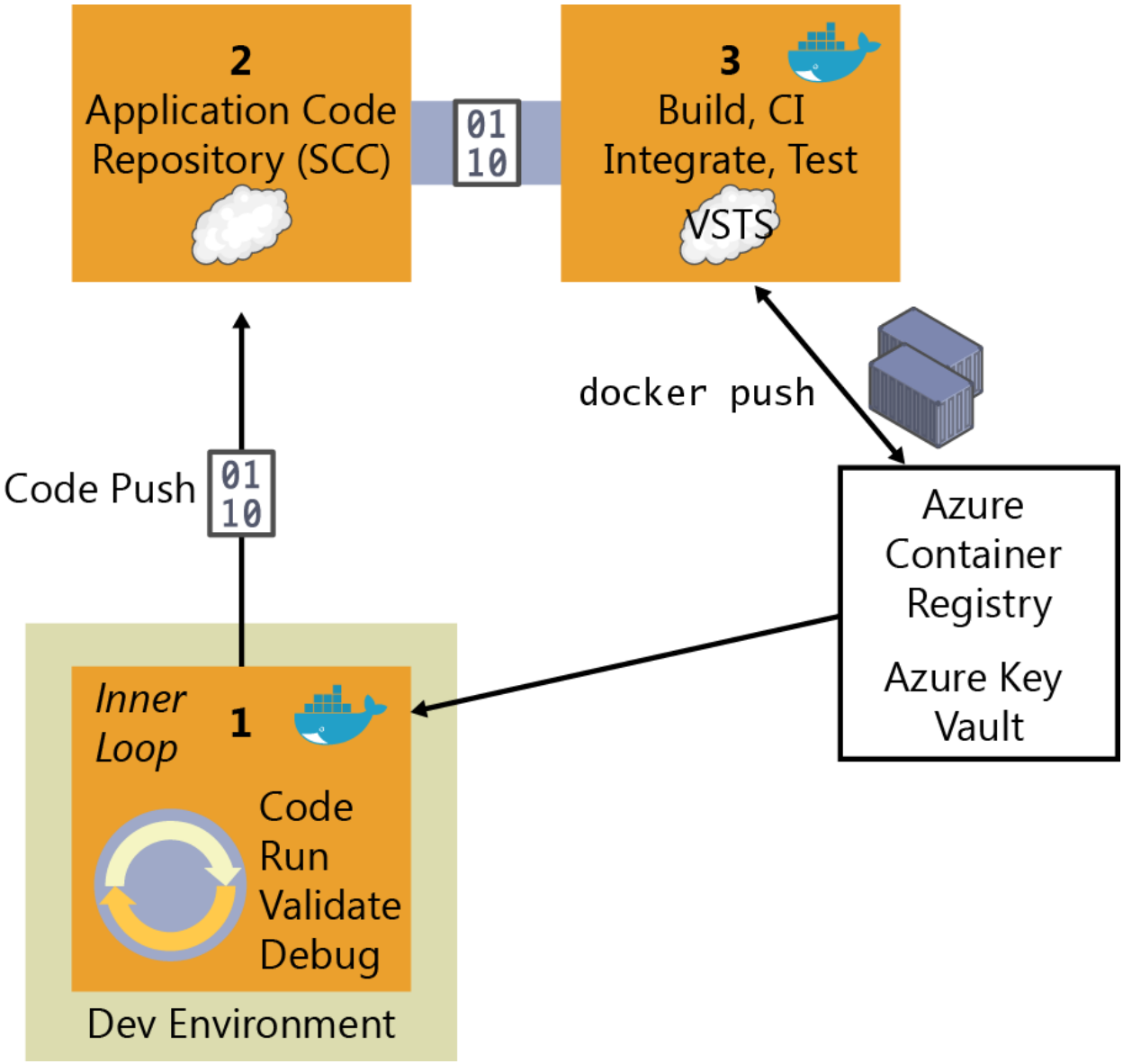


Figura 5-2. Pasos del proceso de CI

Estos son los pasos básicos del flujo de trabajo de CI con Docker y Azure DevOps Services:

1. El desarrollador envía una confirmación a un repositorio de SCC (GIT/Azure DevOps Services, GitHub, etc.).
2. Si usa Azure DevOps Services o GIT, CI estará integrada, lo que significa que es tan fácil como activar una casilla en Azure DevOps Services. Si usa SCC externo (como GitHub), un `webhook` notificará a Azure DevOps Services la actualización o la insertará en GIT/GitHub.
3. Azure DevOps Services extrae el repositorio de SCC, incluido el archivo Dockerfile que describe la imagen, así como el código de la aplicación y la prueba.
4. Azure DevOps Services compila una imagen de Docker y la etiqueta con un número de compilación.
5. Azure DevOps Services crea una instancia del contenedor de Docker en el host de Docker aprovisionado y ejecuta las pruebas adecuadas.
6. Si las pruebas son correctas, primero se vuelve a etiquetar la imagen con un nombre descriptivo para que se sepa que es una "compilación designada" (como "/1.0.0" o cualquier otra etiqueta) y, después, se inserta en el registro de Docker (Docker Hub, Azure Container Registry, DTR, etc.)

Implementación de la canalización de CI con Azure DevOps Services y la extensión de Docker para Azure DevOps Services

Azure DevOps Services de Visual Studio contiene plantillas de compilación y versión que se pueden usar en la canalización de CI/CD para crear imágenes de Docker, insertarlas en un registro autenticado de Docker, ejecutarlas o llevar a cabo otras operaciones que ofrece la CLI de Docker. También agrega una tarea de Docker Compose que se puede usar para compilar, insertar y ejecutar aplicaciones de Docker de varios contenedores, o bien para ejecutar otras operaciones de la CLI de Docker Compose, como se muestra en la figura 5-3.

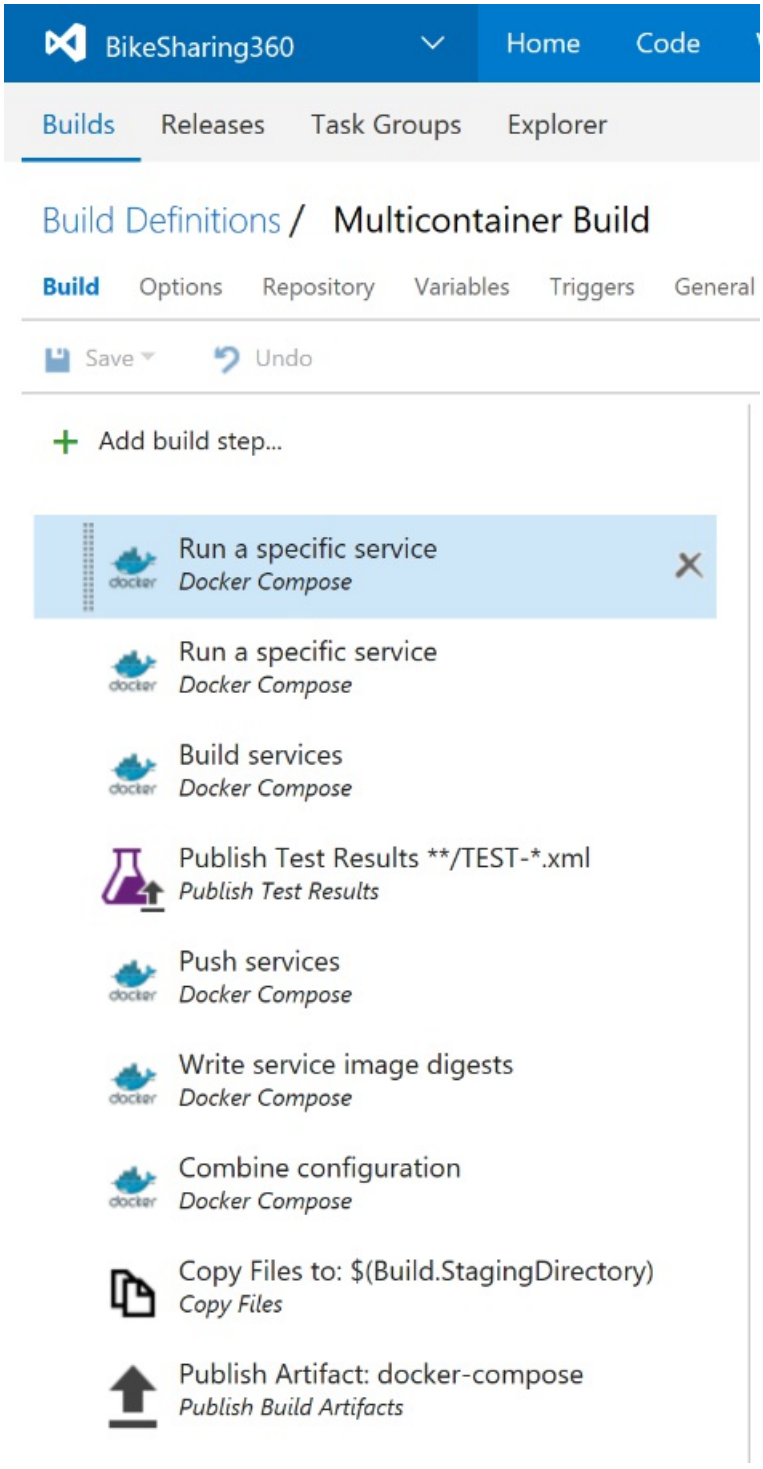


Figura 5-3. Canalización de CI de Docker en Azure DevOps Services, incluidas las plantillas de compilación y versión y tareas asociadas

Puede usar estas plantillas y tareas a fin de construir artefactos de CI/CD para realizar la compilación, prueba e implementación en Azure Service Fabric, Azure Kubernetes Service y ofertas similares.

Con estas tareas de Visual Studio Team Services, un host o máquina virtual de Docker de Linux de compilación aprovisionados en Azure y su registro de Docker preferido (Azure Container Registry, Docker Hub, DTR de Docker privado o cualquier otro registro de Docker), puede ensamblar su canalización de CI de Docker de forma coherente.

** Requisitos: _*

- Azure DevOps Services o, para instalaciones locales, Team Foundation Server 2015 Update 3 o una versión posterior.
- Un agente de Azure DevOps Services que tenga los archivos binarios de Docker.

  Una manera fácil de crear uno de estos agentes consiste en usar Docker para ejecutar un contenedor basado en la imagen de Docker del agente de Azure DevOps Services.

> [INFORMACIÓN] Para obtener más detalles sobre cómo ensamblar una canalización de CI de Docker de Azure DevOps Services y consultar los tutoriales, visite estos sitios:
>
> - Ejecución de un agente de Visual Studio Team Services (ahora Azure DevOps Services) como un contenedor de Docker: https://hub.docker.com/_/microsoft-azure-pipelines-vsts-agent
> - Creación de imágenes de Docker de Linux para .NET con Azure DevOps Services: /archive/blogs/stevelasker/building-net-core-linux-docker-images-with-visual-studio-team-services
> - Creación de una máquina de compilación de Visual Studio Team Services basada en Linux con compatibilidad con Docker: https://www.donovanbrown.com/post/Building-a-Linux-Based-Visual-Studio-Team-Service-Build-Machine-with-Docker-Support

Integración, prueba y validación de aplicaciones de Docker de varios contenedores

Normalmente, la mayoría de las aplicaciones de Docker se componen de varios contenedores, en lugar de uno solo. Un buen ejemplo es una aplicación orientada a microservicios que tenga un contenedor por microservicio. Pero incluso si no se sigue estrictamente una filosofía basada en microservicios, es probable que una aplicación de Docker esté compuesta de varios contenedores o servicios.

Por lo tanto, después de compilar los contenedores de aplicación en la canalización de CI, también deberá implementar, integrar y probar la aplicación en su conjunto con todos sus contenedores en un host de Docker de integración, o incluso en un clúster de prueba en el que se distribuyan los contenedores.

Si usa un solo host, puede recurrir a comandos de Docker como docker-compose para compilar e implementar contenedores relacionados a fin de probar y validar el entorno de Docker en una sola máquina virtual. Aun así, si trabaja con un clúster de orquestador como DC/OS, Kubernetes o Docker Swarm, deberá implementar los contenedores mediante un mecanismo u orquestador diferentes, en función del clúster o programador que haya seleccionado.

A continuación se muestran varios tipos de pruebas que se pueden ejecutar en contenedores de Docker:

- Pruebas unitarias para contenedores de Docker
- Pruebas de grupos de aplicaciones o microservicios interrelacionados
- Pruebas en versiones de producción y de lanzamiento controlado

Lo importante al ejecutar las pruebas funcionales y de integración es que las realice desde fuera de los contenedores. Las pruebas no se incluyen ni se ejecutan en los contenedores que va a implementar, porque los contenedores se basan en imágenes estáticas que deberían ser exactamente iguales a las que implementará en producción.

Una opción práctica al realizar las pruebas en escenarios más avanzados, como cuando se incluyen varios clústeres (clúster de prueba, de almacenamiento provisional y de producción), consiste en publicar las imágenes en un registro, de modo que se pueda probar en varios clústeres.

Inserción de imágenes de Docker de aplicación personalizadas en el registro de Docker global

Una vez que se hayan probado y validado las imágenes de Docker, querrá etiquetarlas y publicarlas en el registro de Docker. El registro de Docker es un elemento fundamental en el ciclo de vida de la aplicación de Docker, ya que es la ubicación central en la que se almacena la prueba personalizada (también conocida como "imagen designada") que se implementará en entornos de producción y control de calidad.

Del mismo modo que el código de la aplicación almacenado en el repositorio de SCC (GIT, etc.) es el "origen fiable", el registro de Docker es el "origen fiable" de la aplicación binaria o los bits que se implementarán en los entornos de producción o control de calidad.

Normalmente, le interesa tener los repositorios privados para las imágenes personalizadas en un repositorio privado en Azure Container Registry, en un registro local como Docker Trusted Registry o en un registro de nube pública con acceso restringido (por ejemplo, Docker Hub), aunque en este último caso, si no se trata de código abierto, debe confiar en la seguridad del proveedor. En cualquier caso, el método que use es similar y se basa en el comando `docker push`, como se muestra en la figura 5-4.

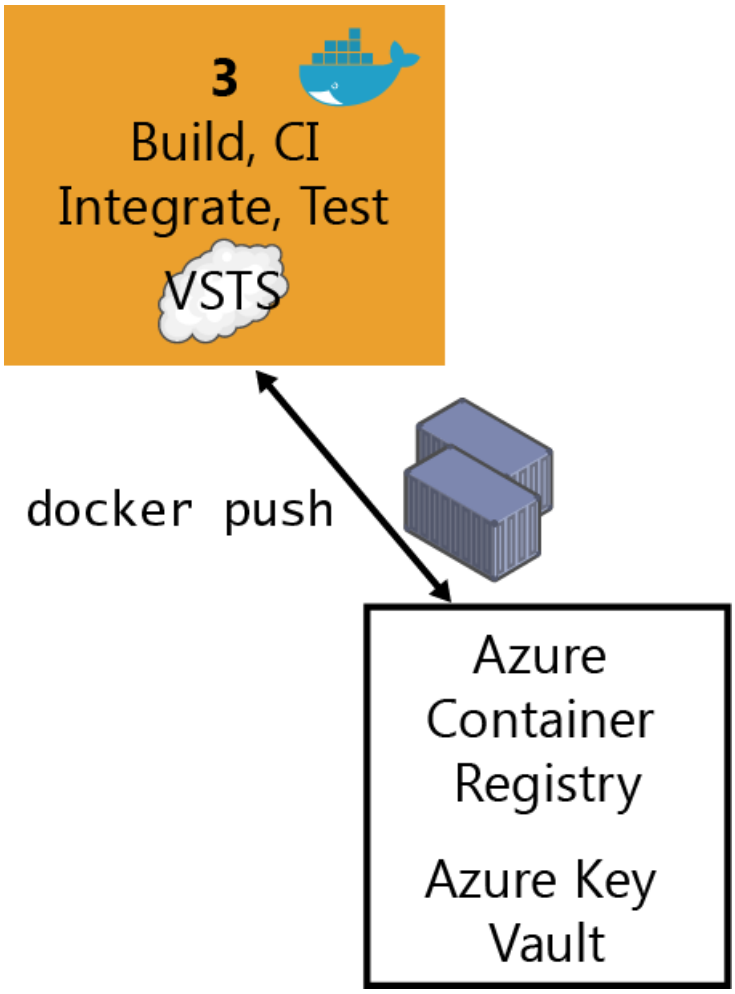


Figura 5-4*. Publicación de imágenes personalizadas en el registro de Docker

En el paso 3, para la compilación, la integración y las pruebas (CI), podría interesarle publicar las imágenes de Docker resultantes en un registro privado o público. Existen proveedores de nube que ofrecen diversos registros de Docker, como Azure Container Registry, Amazon Web Services Container Registry, Google Container Registry, Quay Registry, etc.

Mediante el uso de tareas de Docker, puede insertar un conjunto de imágenes de servicio definidas por un archivo `docker-compose.yml`, con varias etiquetas, en un registro de Docker autenticado (como Azure Container Registry), tal como se muestra en la figura 5-5.

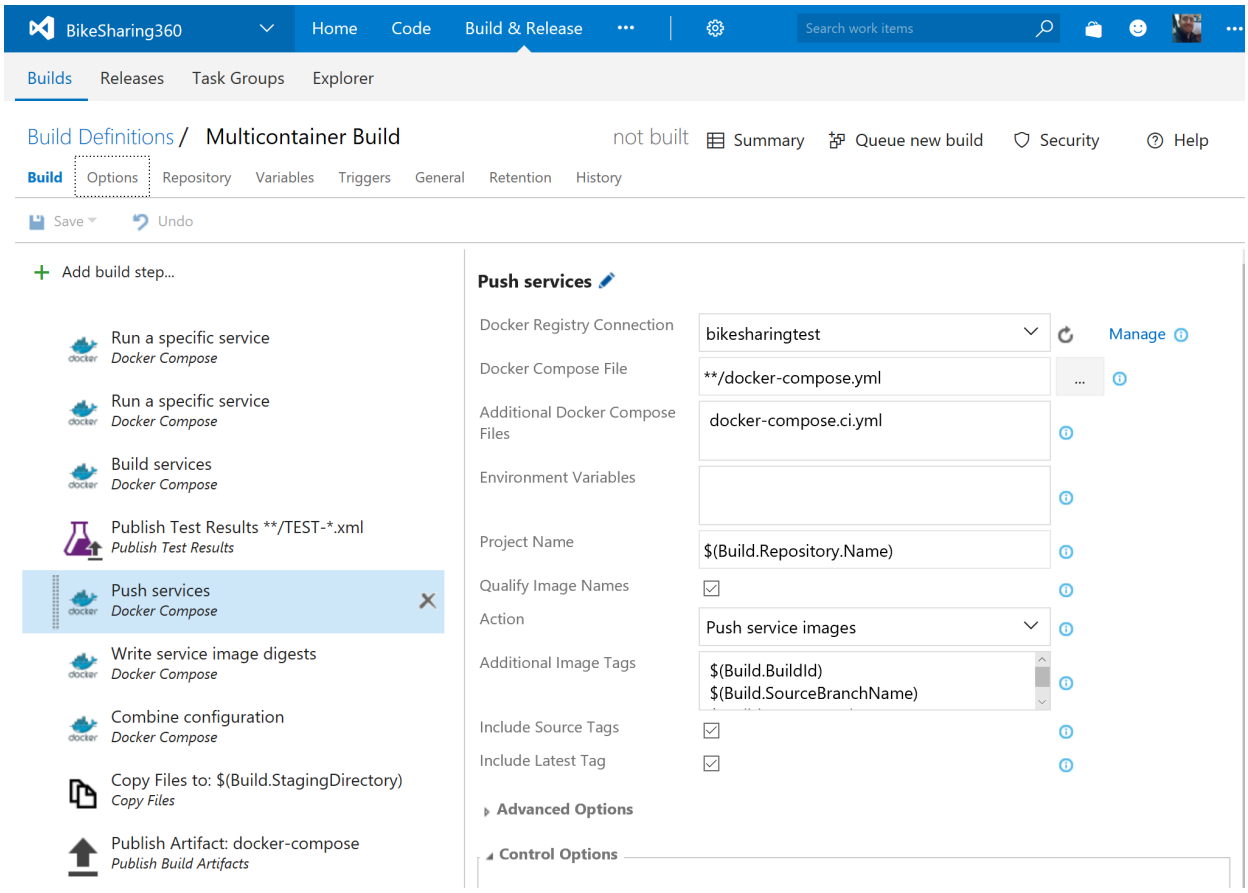


Figura 5-5. Uso de Azure DevOps Services para publicar imágenes personalizadas en un registro de Docker

> [INFORMACIÓN] Para obtener más detalles sobre Azure Container Registry, vea https://aka.ms/azurecontainerregistry.

Paso 4: CD e implementación

La inmutabilidad de las imágenes de Docker garantiza una implementación repetible entre lo que se ha desarrollado, lo que se ha probado a través de CI y lo que se ha ejecutado en producción. Una vez que las imágenes de Docker de la aplicación se han publicado en el registro de Docker (ya sea privado o público), puede implementarlas en los diversos entornos que tenga (producción, control de calidad, almacenamiento provisional, etc.) desde la canalización de CD mediante el uso de tareas de canalización de Azure DevOps Services o Release Management de Azure DevOps Services.

Aun así, este paso depende del tipo de aplicación de Docker que vaya a implementar. La implementación de una aplicación sencilla (desde el punto de vista de la composición y la implementación), por ejemplo, una aplicación monolítica que contenga unos pocos contenedores o servicios y que esté implementada en unos pocos servidores o máquinas virtuales, es diferente de la implementación de una aplicación más compleja, como una aplicación orientada a microservicios con funcionalidades de hiperescala. Estos dos escenarios se explican en las secciones siguientes.

Implementación de aplicaciones compuestas de Docker en varios entornos de Docker

Veamos primero el escenario menos complejo: la implementación en hosts simples de Docker (máquinas virtuales o servidores) en uno o varios entornos (control de calidad, almacenamiento provisional y producción). En este escenario, la canalización de CD puede usar internamente docker-compose (desde las tareas de implementación de Azure DevOps Services) para implementar las aplicaciones de Docker con su conjunto relacionado de contenedores y servicios, como se muestra en la figura 5-6.

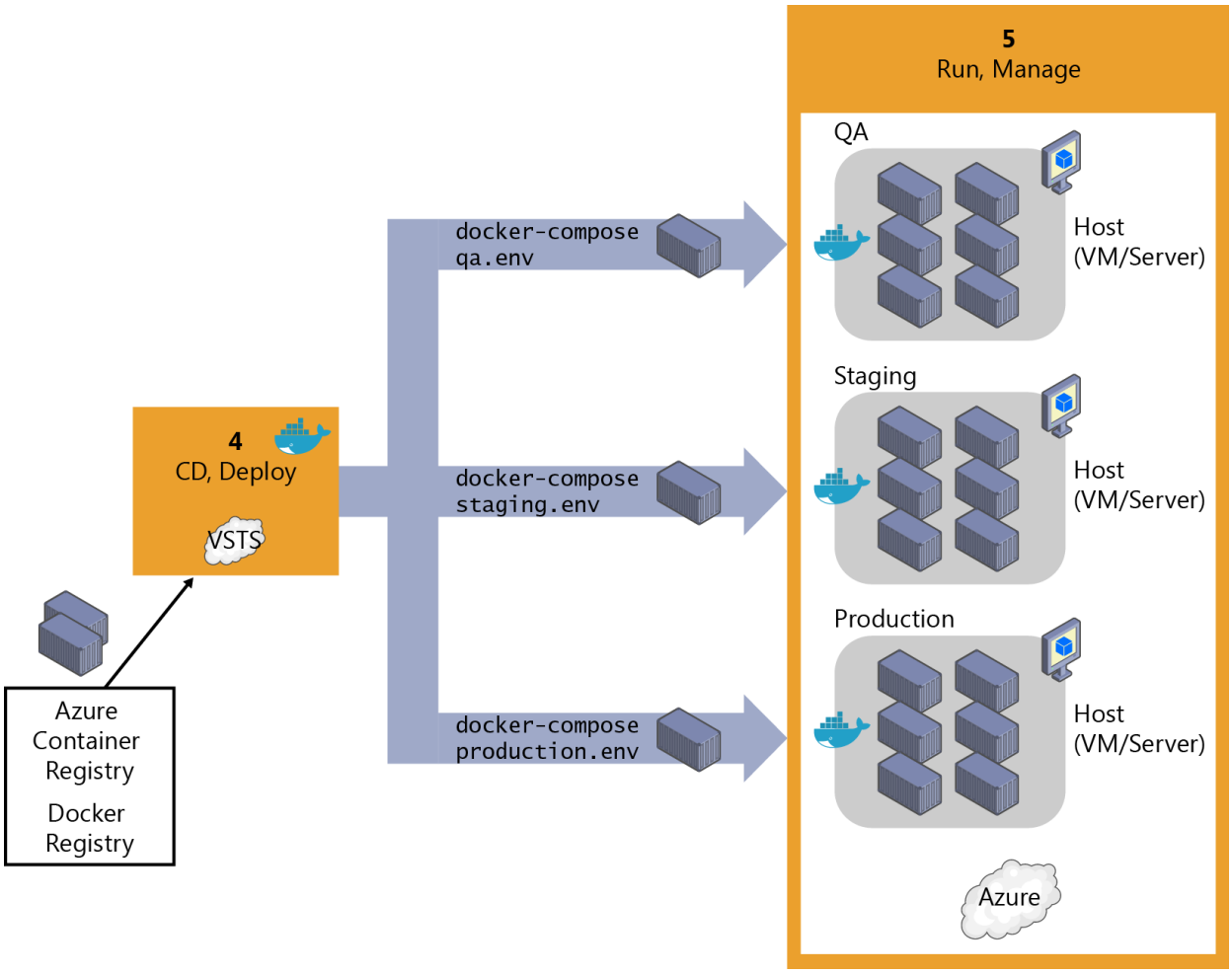


Figura 5-6. Implementación de contenedores de aplicación en el registro de entornos de host de Docker simple

En la figura 5-7 se muestra cómo se pueden conectar la compilación y CI con entornos de control de calidad y pruebas a través de Azure DevOps Services si se hace clic en Docker Compose en el cuadro de diálogo Agregar tarea. Aun así, al realizar la implementación en entornos de almacenamiento provisional o producción, normalmente usará las características de Release Management para administrar varios entornos (como control de calidad, almacenamiento provisional y producción). Si lleva a cabo la implementación en hosts de Docker únicos, lo hará mediante la tarea "Docker Compose" de Azure DevOps Services (que invoca el comando `docker-compose up` en segundo plano). Si va a proceder a la implementación en Azure Kubernetes Service (AKS), usará la tarea de implementación de Docker, como se explica en la sección siguiente.

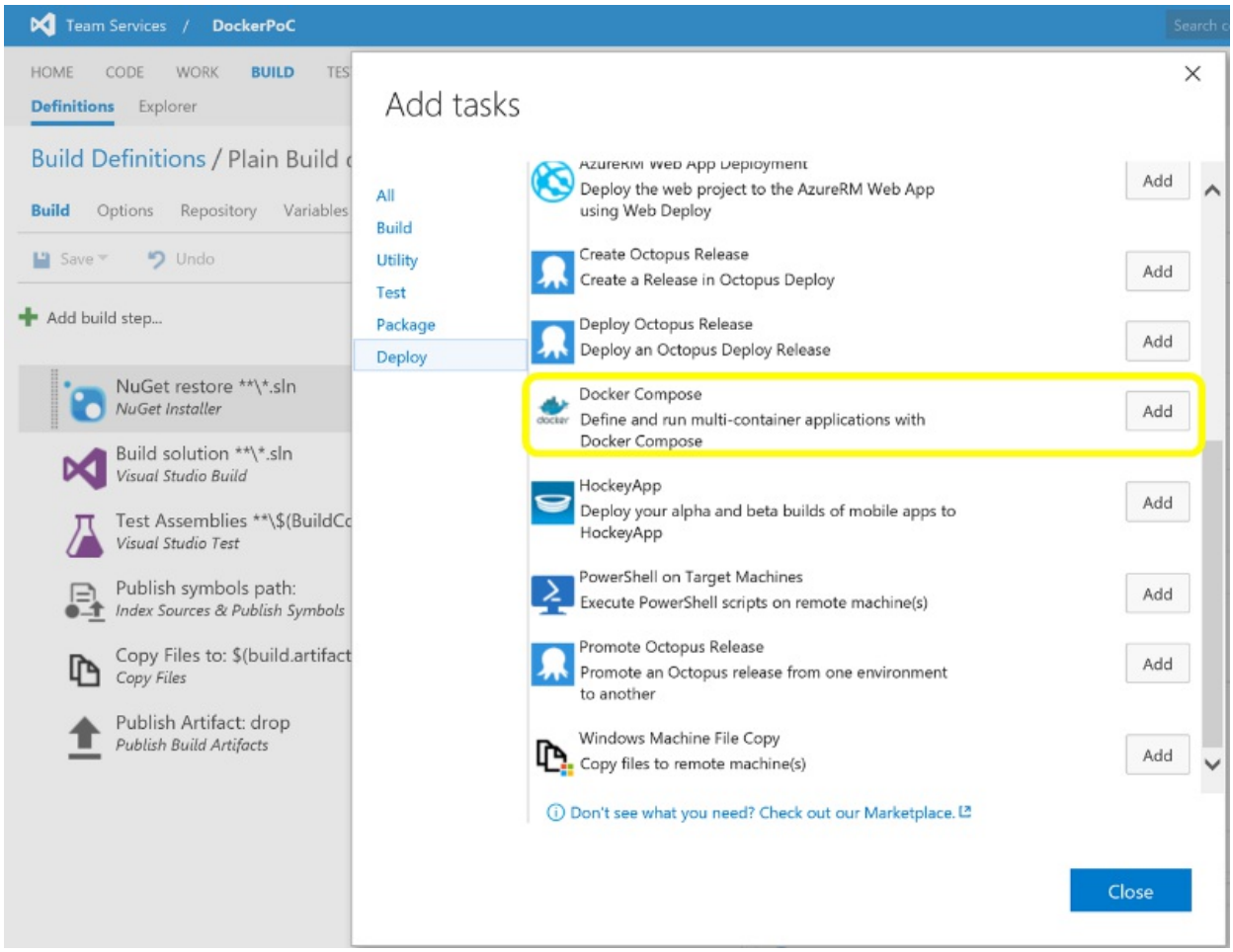


Figura 5-7. Adición de una tarea de Docker Compose en una canalización Azure DevOps Services

Cuando se crea una versión en Azure DevOps Services, se toma un conjunto de artefactos de entrada. Estos artefactos están diseñados para ser inmutables durante la vigencia de la versión y en todos los entornos. Al introducir contenedores, los artefactos de entrada identifican las imágenes de un registro que se van a implementar. Según cómo se identifiquen estas imágenes, no se garantiza que no vayan a cambiar durante la vigencia de la versión. El caso más evidente es cuando se hace referencia a `myimage:latest` desde un archivo `docker-compose`.

Las plantillas de Azure DevOps Services permiten generar artefactos de compilación que contienen resúmenes de las imágenes específicas del registro, con la garantía de que identifican de manera única cada imagen binaria. Esto es lo que realmente le interesa usar como entrada para una versión.

Administración de versiones para entornos de Docker mediante el uso de Release Management de Azure DevOps Services

A través de las plantillas de Azure DevOps Services, puede crear una imagen, publicarla en un registro de Docker, ejecutarla en hosts de Linux o Windows y usar comandos como `docker-compose` para implementar varios contenedores como una aplicación completa, y todo ello gracias a las funcionalidades de Release Management de Azure DevOps Services destinadas a diversos entornos, como se muestra en la figura 5-8.

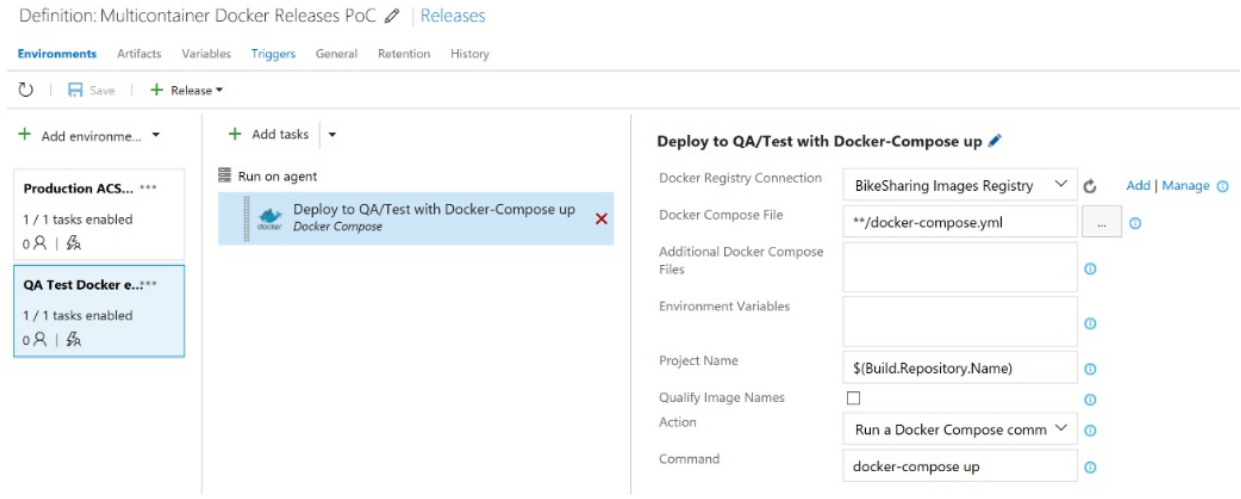


Figura 5-8. Configuración de tareas de Docker Compose de Azure DevOps Services desde Release Management para Azure DevOps Services

Aun así, tenga en cuenta que el escenario que se muestra en la figura 5-6 y que se implementa en la figura 5-8 es sencillo (se implementa en hosts y máquinas virtuales únicos de Docker y solo habrá un contenedor o instancia por imagen) y probablemente solo debería usarse para escenarios de desarrollo o prueba. En la mayoría de los escenarios de producción empresariales, le interesará tener alta disponibilidad y escalabilidad fácil de administrar mediante el equilibrio de carga entre varios nodos, servidores y máquinas virtuales, así como "conmutaciones por error inteligentes" de modo que, si se produce un error en un servidor o un nodo, sus servicios y contenedores se moverán a otro servidor host o máquina virtual. En ese caso, necesita tecnologías más avanzadas, como clústeres de contenedores, orquestadores y programadores. Así pues, la forma de implementar dichos clústeres consiste en controlar los escenarios avanzados que se explican en la sección siguiente.

Implementación de aplicaciones de Docker en clústeres de Docker

La naturaleza de las aplicaciones distribuidas requiere recursos de proceso también distribuidos. Para disponer de funcionalidades a escala de producción, debe tener funcionalidades de agrupación en clústeres que proporcionen alta escalabilidad y alta disponibilidad en función de los recursos agrupados.

Podría implementar contenedores manualmente en esos clústeres desde una herramienta de CLI o una interfaz web, pero debe reservar este tipo de trabajo manual para detectar las pruebas de implementación o con fines de administración, como el escalado horizontal o la supervisión.

Desde el punto de vista de CD, y más concretamente de Azure DevOps Services, puede ejecutar tareas de implementación creadas de forma especial desde los entornos de Release Management de Azure DevOps Services que vayan a implementar las aplicaciones en contenedores en clústeres distribuidos en Container Service, como se muestra en la figura 5-9.

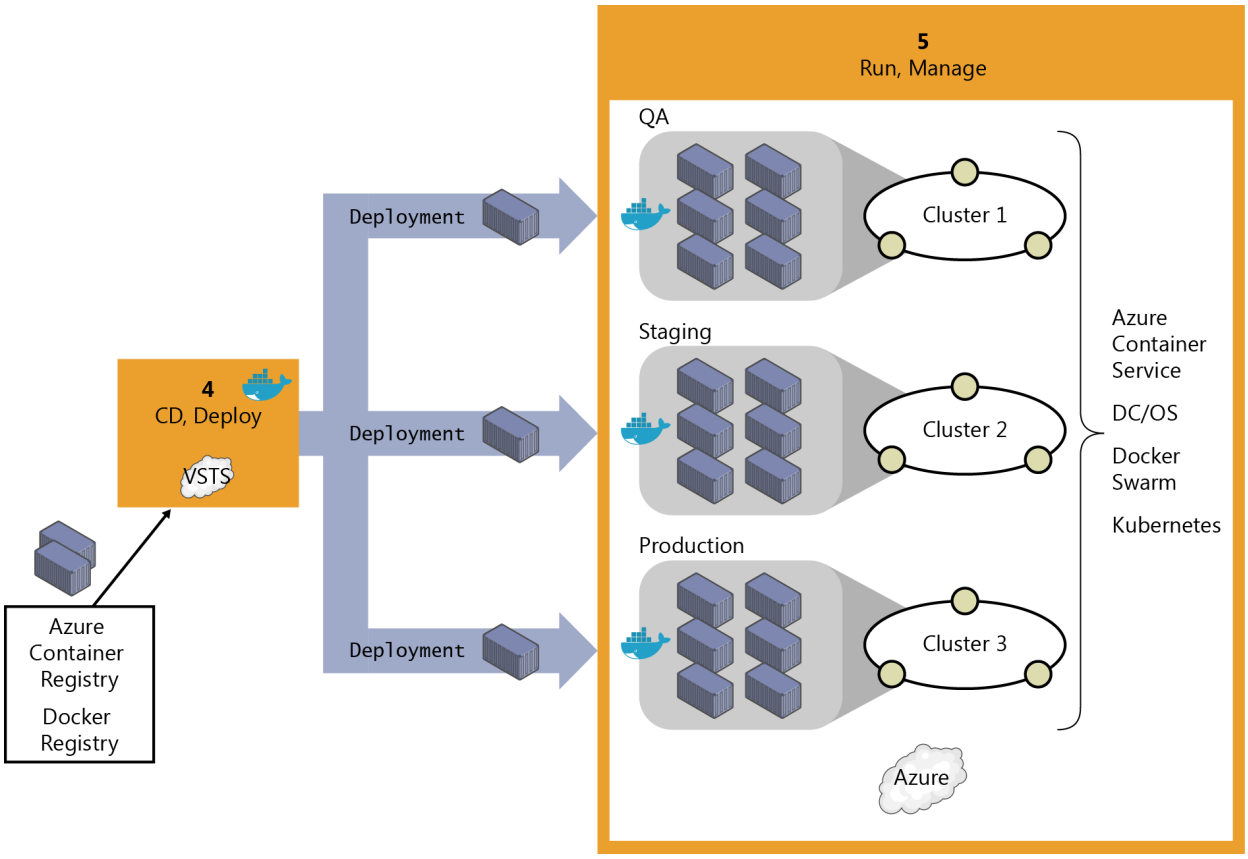


Figura 5-9. Implementación de aplicaciones distribuidas en Container Service

En principio, al implementar en ciertos clústeres u orquestadores, tradicionalmente se usarían mecanismos y scripts de implementación específicos por cada orquestador (es decir, Kubernetes y Service Fabric tienen mecanismos de implementación diferentes), en lugar de la herramienta `docker-compose` (más sencilla y fácil de usar) basada en el archivo de definición `docker-compose.yml`. Aun así, gracias a la tarea de implementación de Docker de Azure DevOps Services, que aparece en la figura 5-10, ahora también puede realizar la implementación en los orquestadores admitidos usando el archivo `docker-compose.yml`, que ya conoce, puesto que la herramienta realiza automáticamente la "traducción" (del archivo `docker-compose.yml` al formato que requiere el orquestador).

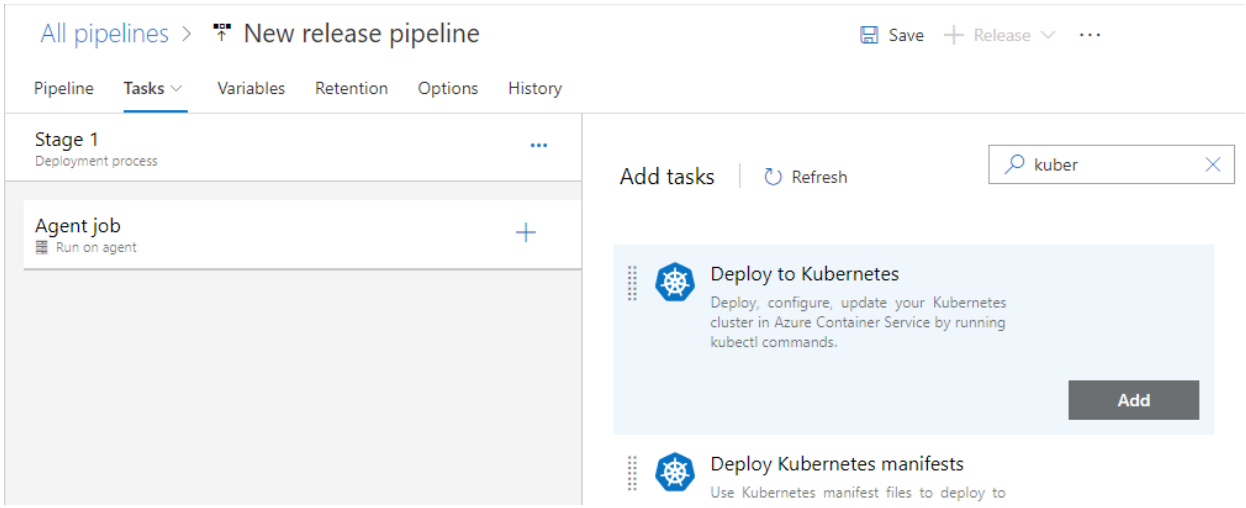


Figura 5-10. Adición de la tarea Implementar en Kubernetes al entorno

En la figura 5-11 se muestra cómo se puede modificar la tarea Implementar en Kubernetes con las secciones disponibles para la configuración. Esta es la tarea que recuperará las imágenes de Docker personalizadas listas para usar que se implementarán como contenedores en el clúster.

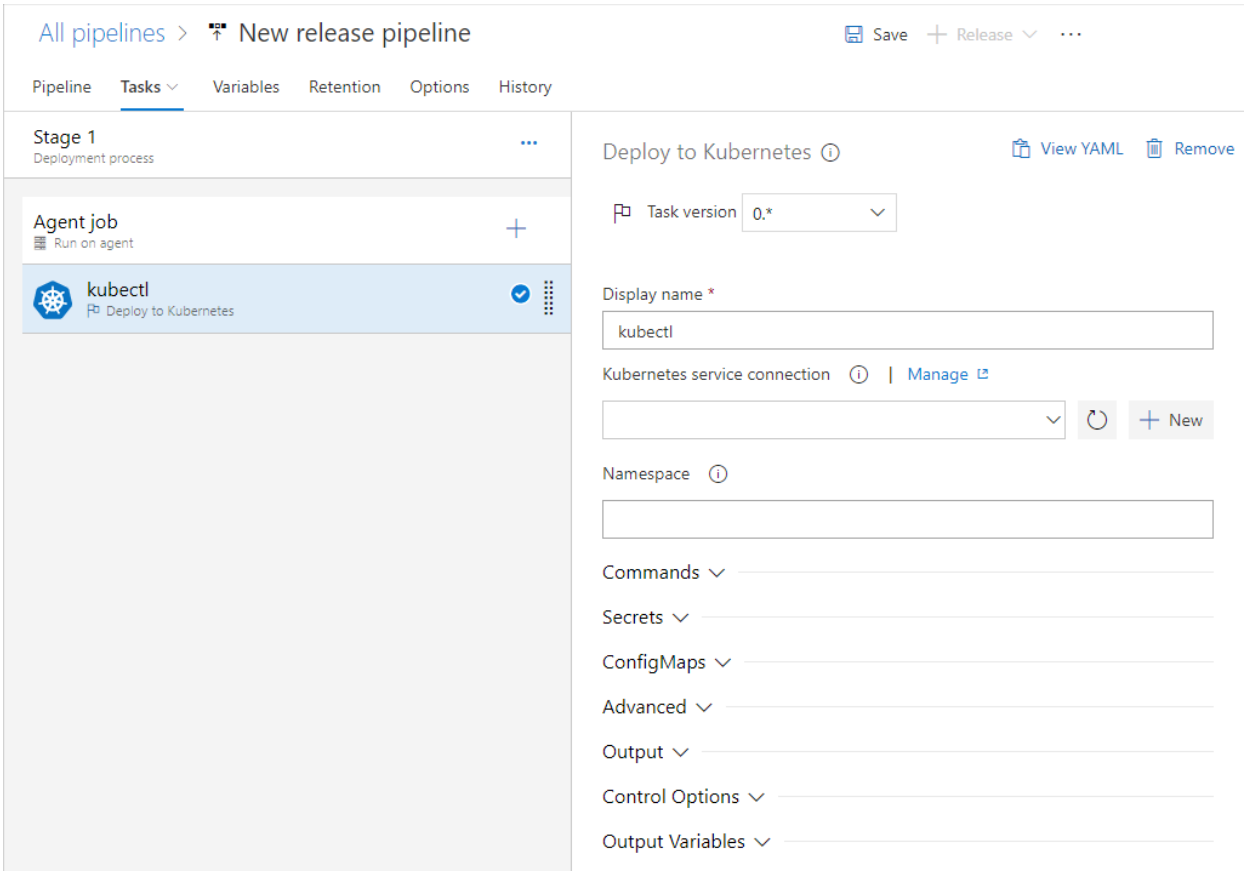


Figura 5-11. Definición de la tarea de implementación de Docker implementada en ACS DC/OS

> [INFORMACIÓN] Para obtener más detalles sobre la canalización de CD con Azure DevOps Services y Docker, visite https://azure.microsoft.com/services/devops/pipelines.

Paso 5: ejecución y administración

Dado que la ejecución y la administración de aplicaciones a nivel de producción empresarial es un asunto de gran importancia por sí mismo, y dado el tipo de operaciones y personas que trabajan en este nivel (operaciones de TI), así como el extenso ámbito de esta cuestión, el próximo capítulo está dedicado al completo a explicarlo.

Paso 6: Supervisión y diagnóstico

Este tema también se trata en el capítulo siguiente como parte de las tareas que lleva a cabo el equipo de TI en los sistemas de producción, pero es importante destacar que los conocimientos obtenidos en este paso deben remitirse al equipo de desarrollo para garantizar una mejora constante de la aplicación. Desde ese punto de vista, también forma parte de DevOps, aunque las tareas y las operaciones suele realizarlas el equipo de TI.

Solo cuando la supervisión y el diagnóstico se encuentran exclusivamente en el ámbito de DevOps la tarea de llevar a cabo los procesos de supervisión y análisis en los entornos beta o de prueba recae en el equipo de desarrollo. Para ello, realizan pruebas de carga o supervisan los entornos beta o de control de calidad en los que se prueban las versiones nuevas.

Creación de canalizaciones de CI/CD en Azure DevOps Services para una aplicación de .NET en contenedores e implementación en un clúster de Kubernetes

06/03/2021 • 2 minutes to read • Edit Online

En la figura 5-12 se puede ver el escenario completo de DevOps, que comprende la administración de código, la compilación de código, la creación de imágenes de Docker, la inserción de imágenes de Docker en un registro de Docker y, por último, la implementación en un clúster de Kubernetes en Azure.

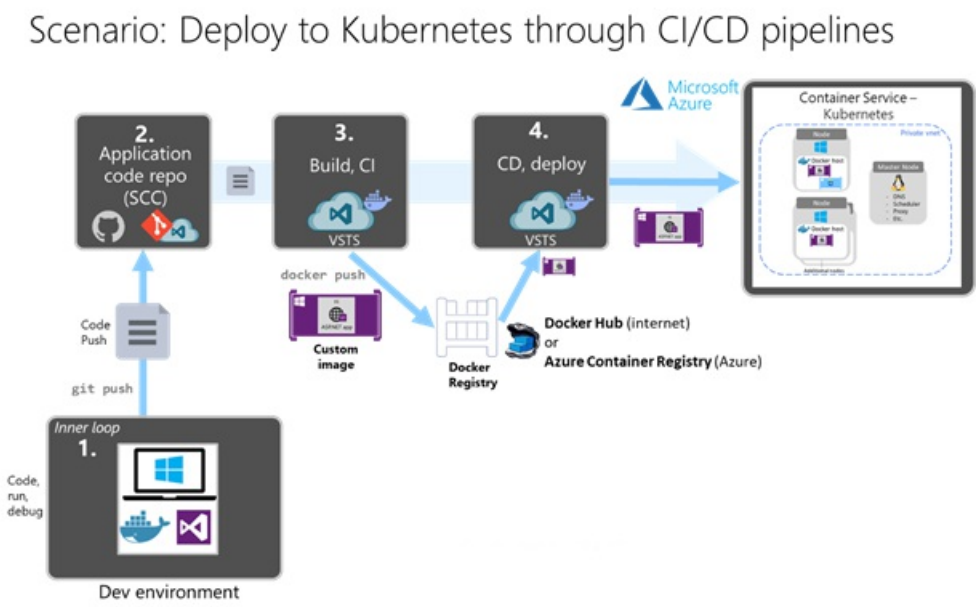


Figura 5-12. Escenario CI/CD de creación de imágenes de Docker y su implementación en un clúster de Kubernetes en Azure

Es importante destacar que las dos canalizaciones, la de compilación/CI y la de versión/CD, se conectan a través del registro de Docker (como Docker Hub o Azure Container Registry). El registro de Docker es una de las principales diferencias con respecto a un proceso tradicional de CI/CD sin Docker.

Como se muestra en la figura 5-13, la primera fase es la canalización de compilación/CI. En Azure DevOps Services se pueden crear canalizaciones de compilación/CI que compilen el código, crear las imágenes de Docker e insertarlas en un registro de Docker como Docker Hub o Azure Container Registry.

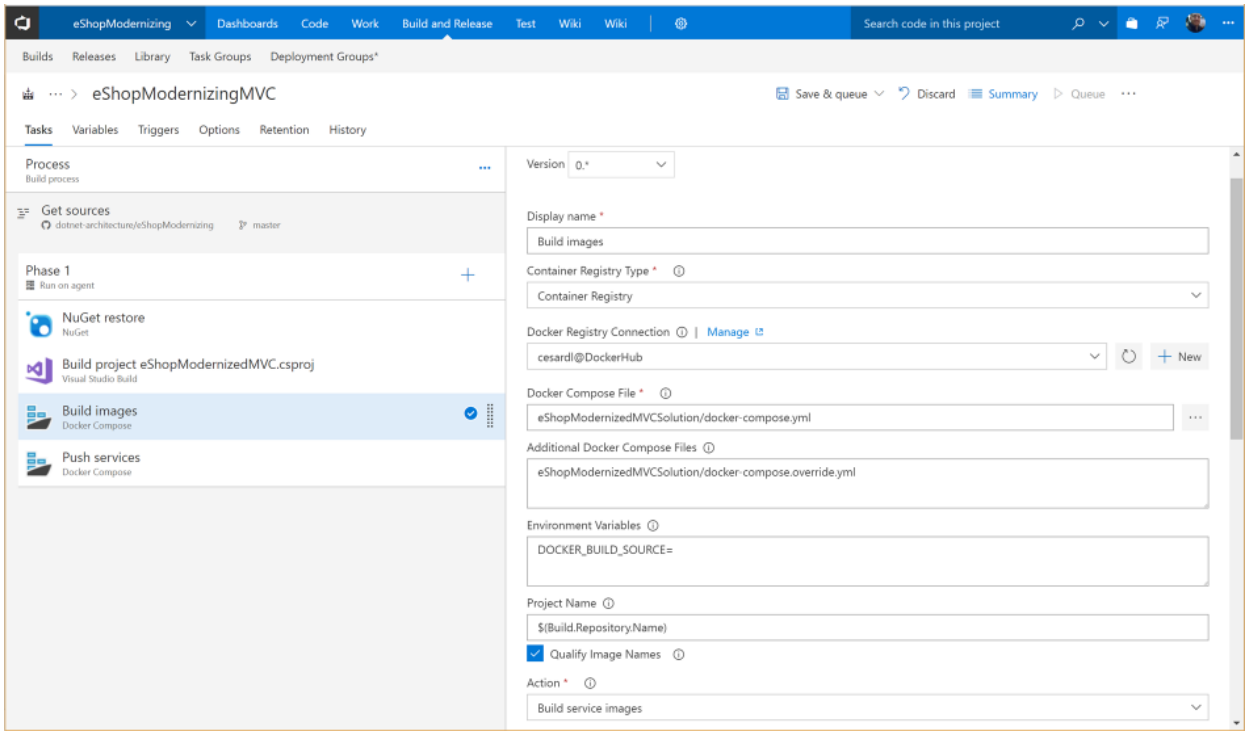


Figura 5-13. Canalización de compilación/CI de Azure DevOps para crear imágenes de Docker e insertar imágenes en un registro de Docker

La segunda fase consiste en la creación de una canalización de implementación o publicación. En Azure DevOps Services se puede crear fácilmente una canalización de implementación destinada a un clúster de Kubernetes mediante tareas de Kubernetes para Azure DevOps Services, tal y como se muestra en la figura 5-14.

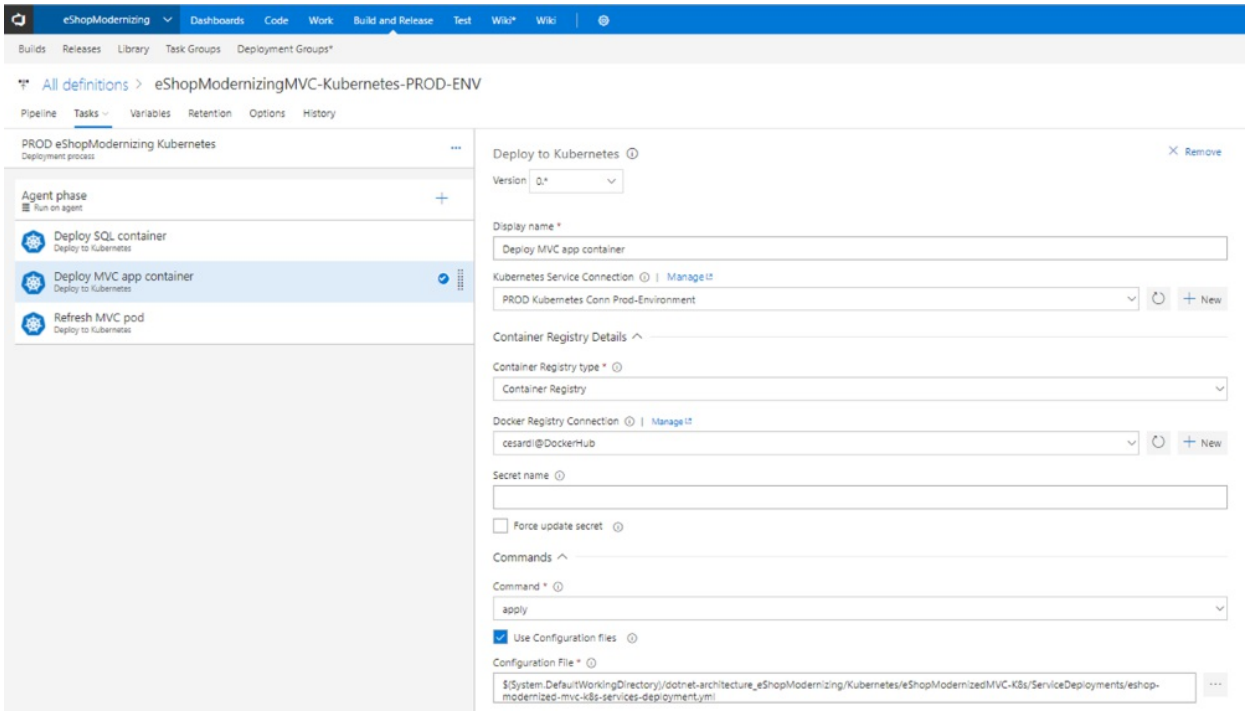


Figura 5-14. Canalización de versión/CD de Azure DevOps Services para implementar en un clúster de Kubernetes

> [Tutorial] Implementación de eShopModernized en Kubernetes:
>
> Para obtener un tutorial detallado de la implementación de canalizaciones de CI/CD de Azure DevOps en Kubernetes, vea esta publicación: https://github.com/dotnet-architecture/eShopModernizing/wiki/04.-How-to-deploy-your-Windows-

Containers-based-apps-into-Kubernetes-in-Azure-Container-Service-(Including-CI-CD)

ANTERIOR SIGUIENTE

Ejecución, administración y supervisión de entornos de producción de Docker

02/11/2020 • 2 minutes to read • Edit Online

Visión: Las aplicaciones empresariales deben ejecutarse con alta disponibilidad y escalabilidad; las operaciones de TI necesitan tener la capacidad de administrar y supervisar los entornos y las aplicaciones propias.

Este último pilar del ciclo de vida de las aplicaciones de Docker en contenedor se centra en cómo ejecutar, administrar y supervisar las aplicaciones en entornos de producción de alta disponibilidad y escalables.

La forma de ejecutar las aplicaciones en contenedor en entornos de producción (tecnologías de plataforma y arquitectura de la infraestructura) está estrechamente relacionada con las plataformas de desarrollo y la arquitectura elegidas analizadas en el capítulo 1 de este libro electrónico y se basa en ellas.

En este capítulo se analizan productos y tecnologías específicos de Microsoft y otros proveedores que puede usar para ejecutar con eficacia aplicaciones distribuidas escalables y de alta disponibilidad, así como para administrarlas y supervisarlas desde la perspectiva de TI.

ANTERIOR SIGUIENTE

Ejecución de aplicaciones basadas en microservicios y compuestas en entornos de producción

02/11/2020 • 4 minutes to read • Edit Online

Las aplicaciones compuestas por varios microservicios deben implementarse en clústeres de orquestador con el fin de simplificar la complejidad de la implementación y hacer que sea viable desde el punto de vista de la TI. Sin un clúster de orchestrator, sería difícil de implementar y escalar horizontalmente una aplicación de microservicios compleja.

Introducción a orquestadores, programadores y clústeres de contenedores

Anteriormente en este libro electrónico, se presentaron los *clústeres* y *programadores* como parte de la discusión sobre arquitectura y desarrollo de software. Kubernetes y Service Fabric son ejemplos de clústeres de Docker. Estos dos orquestadores pueden ejecutarse como parte de la infraestructura proporcionada por Microsoft Azure Kubernetes Service.

Cuando las aplicaciones se escalan horizontalmente en varios sistemas de host, la posibilidad de administrar cada sistema host y abstraer la complejidad de la plataforma subyacente se vuelve atractiva. Eso es precisamente lo que ofrecen los orquestadores y programadores. Vamos a echarles un vistazo aquí:

- **Programadores**. El término "programación" hace referencia a la posibilidad de que un administrador cargue un archivo de servicio en un sistema host que establezca cómo ejecutar un contenedor específico. El inicio de contenedores en un clúster de Docker suele conocerse como programación. Aunque la programación se refiere al acto específico de cargar la definición de servicio, en un sentido más general, los programadores son responsables de conectarse al sistema de inicio de un host para administrar los servicios en la medida en que sea necesario.

  Un programador de clústeres tiene varios objetivos: usar de forma eficaz los recursos del clúster, trabajar con restricciones de selección de ubicación proporcionadas por el usuario, programar aplicaciones rápidamente para no dejarlas en un estado pendiente, tener un grado de "imparcialidad", ser sistemáticos ante los errores y estar siempre disponible.

- **Orquestadores**. Las plataformas amplían las funcionalidades de administración del ciclo de vida a cargas de trabajo complejas y de varios contenedores, implementadas en un clúster de hosts. Mediante la abstracción de la infraestructura del host, las herramientas de orquestación ofrecen a los usuarios una forma de tratar todo el clúster como un único destino de implementación.

  El proceso de orquestación implica herramientas y una plataforma que pueda automatizar todos los aspectos de la administración de aplicaciones desde la selección de ubicación o implementación iniciales por contenedor; el traslado de contenedores a diferentes hosts según el mantenimiento o rendimiento del host; el control de versiones y actualizaciones graduales, y funciones de supervisión del mantenimiento que admitan escalabilidad y conmutación por error, y mucho más.

  Orquestación es un término amplio que hace referencia a la programación de contenedores, la administración del clúster y, posiblemente, el aprovisionamiento de hosts adicionales.

Las funcionalidades proporcionadas por los orquestadores y los programadores son difíciles de desarrollar y crear desde cero, por lo tanto, normalmente preferirá usar soluciones de orquestación ofrecidas por proveedores.

ANTERIOR SIGUIENTE

Administración de entornos de Docker en producción

02/11/2020 • 5 minutes to read • Edit Online

El proceso de administrar y orquestar un clúster consiste en controlar un grupo de hosts. Esto puede conllevar agregar y eliminar hosts de un clúster, obtener información sobre el estado actual de los hosts y los contenedores, e iniciar y detener los procesos. La administración y la orquestación del clúster están estrechamente relacionadas con la programación, dado que el programador debe tener acceso a cada host del clúster a fin de programar servicios. Por este motivo, suele usarse la misma herramienta para ambos fines.

Servicio de contenedor y herramientas de administración

El servicio de contenedor proporciona una rápida implementación de agrupación en clústeres de contenedor de código abierto y soluciones de orquestación populares. Usa imágenes de Docker para asegurarse de que los contenedores de la aplicación sean totalmente portátiles. Mediante el servicio de contenedor, puede implementar clústeres de DC/OS (con tecnología de Mesosphere y Apache Mesos) y Docker Swarm con plantillas de Azure Resource Manager o Azure Portal para asegurarse de poder escalar estas aplicaciones a miles de contenedores, o incluso a decenas de miles.

Estos clústeres se implementan mediante Conjuntos de escala de máquinas virtuales de Azure y sacan provecho de las posibilidades que ofrecen las funciones de red y almacenamiento de Azure. Para acceder al servicio de contenedor, necesita una suscripción de Azure. Con el servicio de contenedor, puede sacar provecho de las características de nivel empresarial de Azure manteniendo la portabilidad de la aplicación, incluso en las capas de orquestación.

En la tabla 6-1 se enumeran las herramientas de administración comunes relacionadas con los orquestadores, los programadores y la plataforma de agrupación en clústeres.

Tabla 6-1. Herramientas de administración de Docker

HERRAMIENTAS DE ADMINISTRACIÓN	DESCRIPTION	ORQUESTADORES RELACIONADOS
Azure Monitor para contenedores	Herramienta de administración de Kubernetes dedicada a Azure.	Azure Kubernetes Services (AKS)
Interfaz de usuario web de Kubernetes (panel)	Herramienta de administración de Kubernetes que puede supervisar y administrar un clúster local de Kubernetes.	Azure Kubernetes Service (AKS) Kubernetes local
Azure Portal para Service Fabric Azure Service Fabric Explorer	Versión de escritorio y en línea para administrar clústeres de Service Fabric en Azure, en el entorno local, en el desarrollo local y en otras nubes.	Azure Service Fabric
Supervisión de contenedores (Azure Monitor)	Administración de contenedores general y solución de supervisión. Puede administrar clústeres de Kubernetes mediante Azure Monitor para contenedores.	Azure Service Fabric Azure Kubernetes Service (AKS) Mesosphere DC/OS y otros

Azure Service Fabric

Otra opción para la administración y la implementación del clúster es Azure Service Fabric. Service Fabric es una plataforma de microservicios de Microsoft que incluye la orquestación de contenedores, así como modelos de programación para desarrolladores que permiten compilar aplicaciones de microservicios altamente escalables. Service Fabric es compatible con Docker en contenedores de Windows y Linux y se puede ejecutar en servidores de Windows y Linux.

A continuación se enumeran herramientas de administración de Service Fabric:

- Azure Portal para Service Fabric permite llevar a cabo en un clúster operaciones relacionadas con este (crear, actualizar o eliminar) o configurar su infraestructura (máquinas virtuales, equilibrador de carga, redes, etc.).
- Azure Service Fabric Explorer es una herramienta multiplataforma de escritorio e interfaz de usuario web especializada que proporciona información y permite realizar ciertas operaciones en el clúster de Service Fabric, desde el punto de vista de los nodos o las máquinas virtuales y desde el punto de vista de la aplicación y los servicios.

Supervisión de servicios de aplicaciones en contenedor

02/11/2020 • 3 minutes to read • Edit Online

Es fundamental que las aplicaciones que se dividen en varios contenedores y microservicios cuenten con una forma de supervisar y analizar el comportamiento de toda la aplicación.

Azure Monitor

Azure Monitor es un servicio de análisis extensible que supervisa su aplicación activa. Le ayuda a detectar y diagnosticar problemas de rendimiento y a comprender qué hacen los usuarios realmente con su aplicación. Está diseñado para desarrolladores, con la intención de ayudarle a mejorar continuamente el rendimiento y la facilidad de uso de sus servicios o aplicaciones. Azure Monitor trabaja tanto con web/servicios como con aplicaciones independientes en una amplia variedad de plataformas como.NET, Java, Node.js y muchas otras, hospedadas en local o en la nube.

Recursos adicionales

- **Información general sobre Azure Monitor**
  /azure/azure-monitor/overview
- **¿Qué es Application Insights?**
  /azure/azure-monitor/app/app-insights-overview
- **¿Qué es el Explorador de métricas de Azure Monitor?**
  /azure/azure-monitor/platform/data-platform-metrics
- **Solución de supervisión de contenedores de Azure Monitor**
  /azure/azure-monitor/insights/containers

Servicios de seguridad y copia de seguridad

Hay muchas tareas de soporte técnico con muchos detalles que hay que controlar para garantizar que las aplicaciones y la infraestructura estén en condiciones óptimas para satisfacer las necesidades de la empresa. Esta situación se complica en el ámbito de los microservicios, por lo que ha de tener vistas generales y detalladas cuando tenga que tomar medidas.

Azure cuenta con las herramientas necesarias para administrar y proporcionar una vista unificada de cuatro aspectos críticos tanto de los recursos en la nube como en local:

- **Seguridad**. Con Azure Security Center.
  - Obtenga total visibilidad y control sobre la seguridad de sus máquinas virtuales, aplicaciones y cargas de trabajo.
  - Centralice la administración de las directivas de seguridad e integre herramientas y procesos existentes.
  - Detecte amenazas reales con análisis avanzado.
- **Copia de seguridad**. Con Azure Backup.
  - Evite costosas interrupciones de actividad de la empresa, cumpla los objetivos de cumplimiento normativo y proteja sus datos frente a ransomware y errores humanos.
  - Mantenga los datos de copia de seguridad cifrados en tránsito y en reposo.

    - Garantice el acceso basado en la autenticación multifactor para evitar el uso no autorizado.
- **Recursos locales**. Con soluciones de nube híbrida.

ANTERIOR SIGUIENTE

Puntos clave

02/11/2020 • 2 minutes to read • Edit Online

- Las soluciones basadas en contenedores ofrecen importantes ventajas de ahorro de costos porque los contenedores resuelven los problemas de implementación causados por los errores de dependencia en los entornos de producción, por lo que se mejoran considerablemente las operaciones de producción y de DevOps.
- Docker se ha convertido en el estándar de facto del sector de los contenedores y es compatible con los proveedores más importantes de los ecosistemas Linux y Windows, incluido Microsoft. En el futuro, Docker estará omnipresente en todos los centros de datos en la nube o locales.
- Un contenedor de Docker se está convirtiendo en la unidad de implementación estándar para cualquier aplicación o servicio basados en servidor.
- Los orquestadores de Docker, como los que se proporcionan en Azure Kubernetes Service (AKS) y Azure Service Fabric, son fundamentales e indispensables para cualquier aplicación de varios contenedores o basada en microservicios con importantes requisitos de complejidad y escalabilidad.
- Un entorno de DevOps de un extremo a otro compatible con la integración continua/implementación continua (CI/CD) y que se conecte a entornos de Docker en producción proporciona agilidad y, en última instancia, mejora el tiempo de comercialización de las aplicaciones.
- Azure DevOps Services simplifica en gran medida el entorno de DevOps mediante la implementación en entornos de Docker desde las canalizaciones de CI/CD. Esto es cierto tanto en los entornos de Docker sencillos como en orquestadores de contenedores y microservicios avanzados basados en Azure.

PREVIOUS