MÓDULO 9

Gestión de compilaciones de Red Hat OpenShift

EL PROCESO DE COMPILACIONES DE RED HAT OPENSHIFT

PROCESO DE COMPILACIÓN

- Un proceso de compilación usa parámetros de entrada y código fuente o binarios de la aplicación para crear imágenes de contenedor.

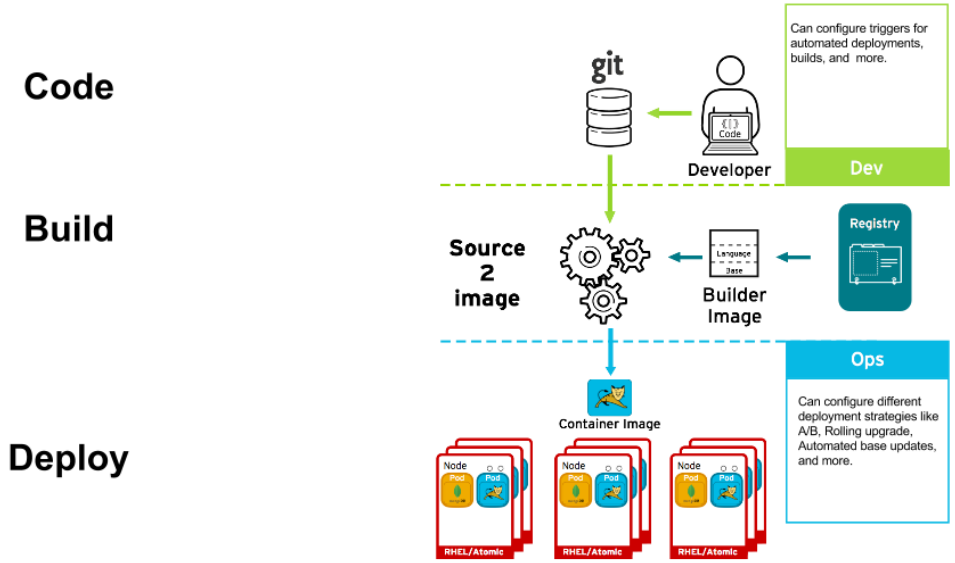
- El recurso BuildConfig contiene la definición del proceso de compilación, que incluye lo siguiente

  - Un desencadenador de compilación que inicia el proceso de compilación.

  - Una estrategia de compilación que define cómo funciona el proceso de compilación.

  - Una o más fuentes de entrada, como definiciones git, binarias o en línea.

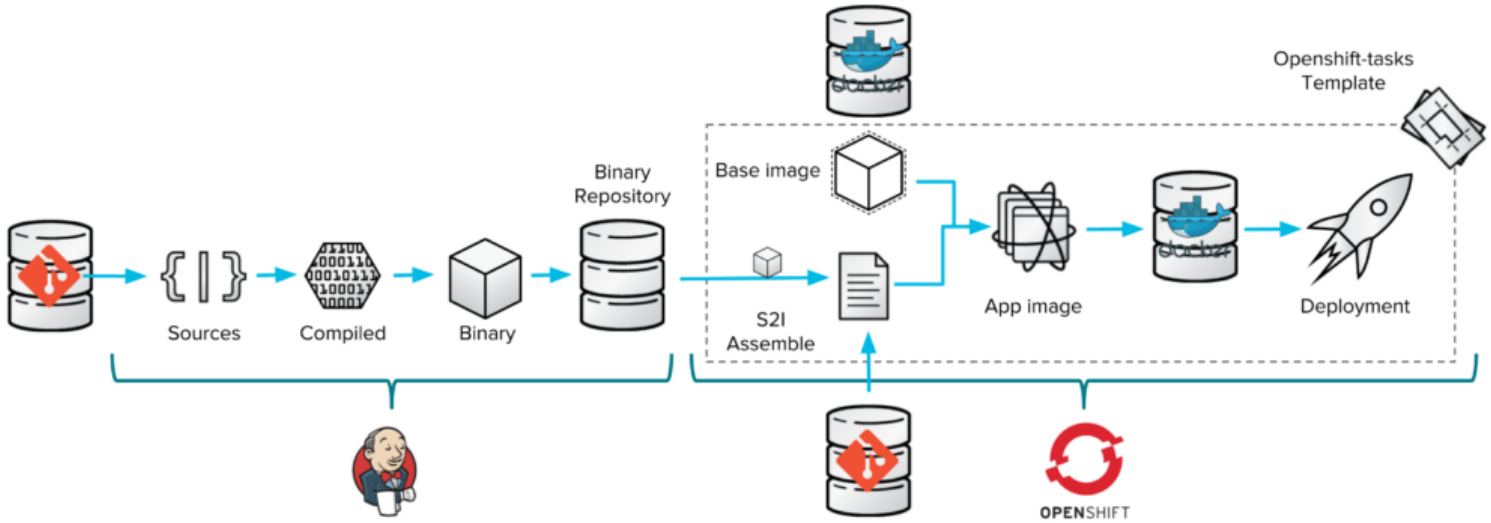
  - La salida del proceso de compilación, que suele ser una imagen de contenedor ejecutable.



ESTRATEGIAS DE COMPILACIÓN

Las siguientes son las estrategias de compilación disponibles en OpenShift:

- Compilación de fuente a imagen (S2I)
- Compilación de Docker
- Compilación personalizada



RECURSO BUILDCONFIG

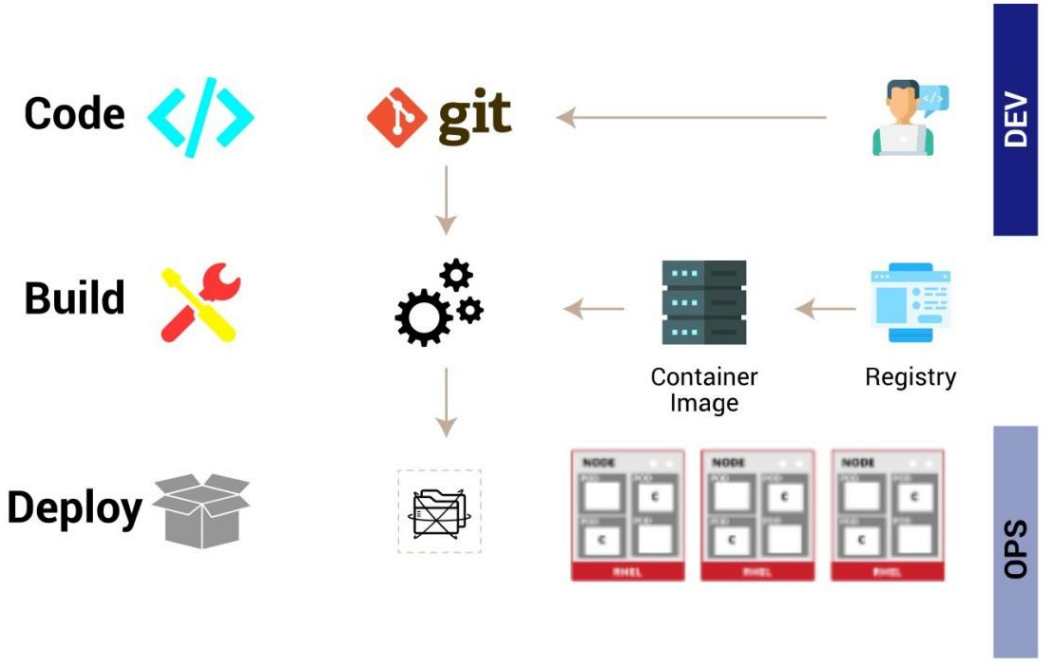
- OpenShift genera un BuildConfig cuando crea una aplicación ya sea ejecutando los comandos oc new-app o oc new-build, o bien usando la consola web.
- Puede editar la definición BuildConfig para personalizarla según sus necesidades.
- El siguiente ejemplo compila una aplicación PHP mediante la estrategia source y una fuente de entrada git:

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "php-sample-build"
spec:
  runPolicy: "Serial" (1)
  triggers: (2)
    - type: "ImageChange"
  source: (3)
    git:
      uri: "http://services.lab.example.com/php-helloworld"
  strategy: (4)
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "ruby-20-centos7:latest"
  output: (5)
    to:
      kind: "ImageStreamTag" (6)
      name: "origin-ruby-sample:latest"
```

1. El atributo `runPolicy` define si varias compilaciones se pueden ejecutar simultáneamente. El valor `Serial` representa que solo se puede ejecutar una compilación a la vez.
2. El desencadenador `ImageChange` que crea una nueva compilación cuando cambia el flujo de imágenes `ruby-20-centos7:latest`.
3. El atributo de git `source` es responsable de definir la fuente de entrada de la compilación. Un `BuildConfig` puede tener varias entradas.
4. Define la estrategia de compilación `Source`, que utiliza la tecnología S2I para compilar la imagen de contenedor.
5. El atributo `output` define dónde se envía la nueva imagen de contenedor después de una compilación correcta.
6. Tipo de imagen de salida establecido en `ImageStreamTag`, que dirige la creación de la imagen al registro de imágenes integrado en OpenShift. Para especificar cualquier registro de imágenes, use el tipo `DockerImage`, que permite nombres de imágenes completamente calificados.

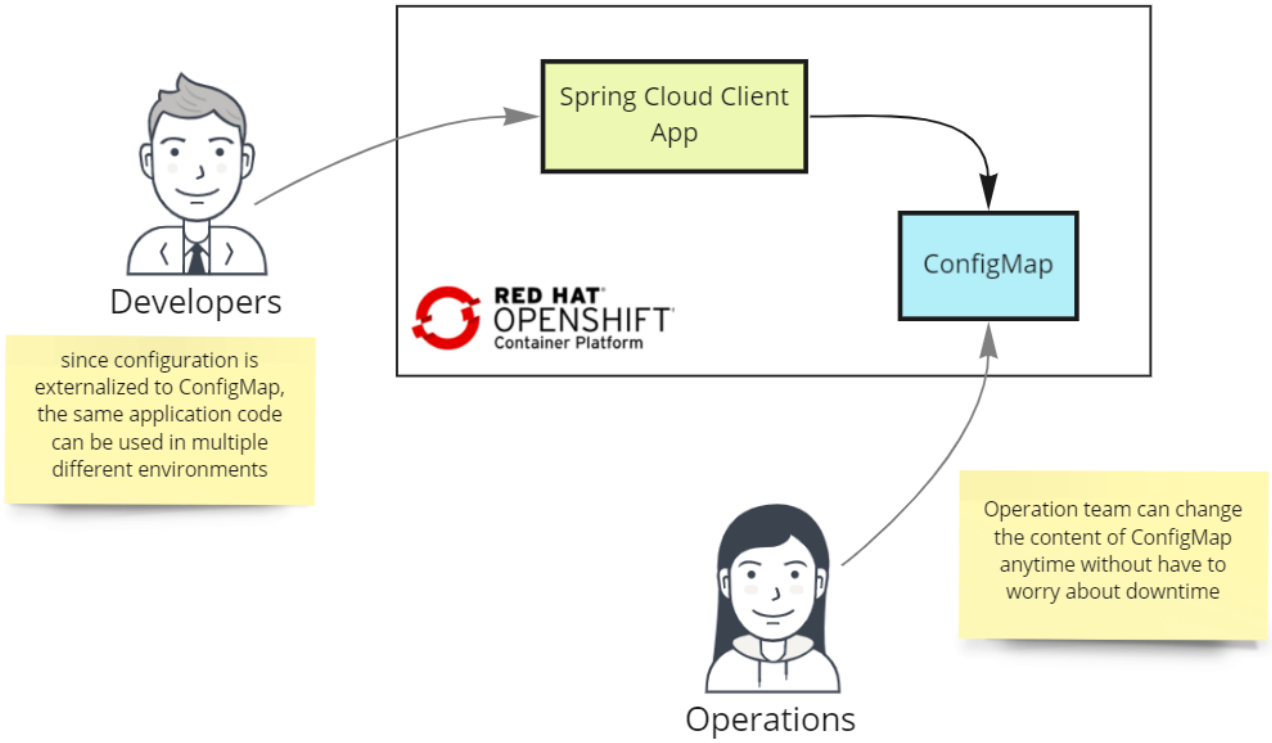
ENTRADA DE COMPILACIÓN

- Una entrada de compilación proporciona contenido de origen para las compilaciones.
- OpenShift soporta los siguientes tipos de fuentes de entrada, enumerados en orden de prioridad:
    - Inline Dockerfile
    - Image
    - Git
    - InputScrets
    - Externatl artifacts



USO DE INFORMACIÓN CONFIDENCIAL EN COMPILACIONES

- Las compilaciones pueden requerir información confidencial, como credenciales, tokens o certificados, para acceder a las dependencias de compilación.
- Para evitar que aparezca información confidencial en la imagen de salida, almacene los datos confidenciales en los configmaps y los secrets de Kubernetes.



EJEMPLO DE CONFIGMAPS Y SECRETS

- El siguiente ejemplo muestra cómo proporcionar de forma segura una compilación S2I para una aplicación Java con información de acceso para descargar dependencias externas:

```
...
source:
  git:
    uri: "http://services.lab.example.com/java-helloworld"
  configMaps:
    - configMap: (1)
        name: settings-mvn
      destinationDir: ".m2"
  secrets:
    - secret: (2)
        name: secret-mvn
      destinationDir: ".ssh"
```

(1) Mapa de configuración que contiene el archivo `settings.xml` y lo monta en el directorio `.m2` del contenedor de la compilación.

(2) Secreto que monta la clave ssh en el directorio `.ssh` del contenedor de la compilación.

INFORMACION DE ACCESO SEGURO

- El siguiente ejemplo muestra cómo proporcionar información de acceso de forma segura para descargar dependencias para la estrategia Docker y el lenguaje de programación Java:

```
spec:
  dockerStrategy:
    volumes:
      - name: settings-mvn
        mounts:
        - destinationPath: /opt/app-root/src/.m2
        source:
          type: ConfigMap (1)
          configMap:
            name: my-config
      - name: secret-mvn (2)
        mounts:
        - destinationPath: /opt/app-root/src/.ssh
        source:
          type: Secret
          secret:
            secretName: my-secret
```

(1) Compilar volumen con un mapa de configuración que contiene `settings.xml` y lo monta en el directorio `/opt/app-root/src/.m2`.

(2) Compilar volumen con un secreto que contiene la clave ssh y lo monta en el directorio `/opt/app-root/src/.ssh`.

DETECCIÓN DE LENGUAJE DE FUENTE A IMAGEN (S2I)

- La característica de detección del lenguaje de programación se basa en buscar nombres de archivos específicos en la raíz del repositorio de git.

- En la siguiente tabla se muestran algunas opciones comunes, pero no es una extensa lista de todos los lenguajes de fuente a imagen compatibles:

Archivos	Compilador de lenguaje	Lenguaje de programación
`Dockerfile`	N/D	Compilación de Dockerfile (no S2I)
`pom.xml`	`jee`	Java (con JBoss EAP)
`app.json,package.json`	`nodejs`	Node.js (JavaScript)
`composer.json,index.php`	`php`	PHP

LENGUAJE DE FUENTE DE IMAGEN

- OpenShift sigue un algoritmo de varios pasos para determinar si la URL apunta a un repositorio de código fuente y, si ese es el caso, cuál es la imagen de compilador que debe realizar la compilación.

- La siguiente es una descripción simplificada del proceso S2I:

  1.- Si la URL apunta a un repositorio de git, OpenShift recupera una lista de archivos del repositorio y busca un archivo Dockerfile.

  2.- OpenShift busca flujos de imágenes que coincidan con el nombre del compilador como el valor de la anotación supports

  3.- Si ninguna anotación coincide, OpenShift busca un flujo de imágenes cuyo nombre coincida con el nombre del compilador de lenguaje

FORZAR U OMITIR FLUJO DE IMAGENES

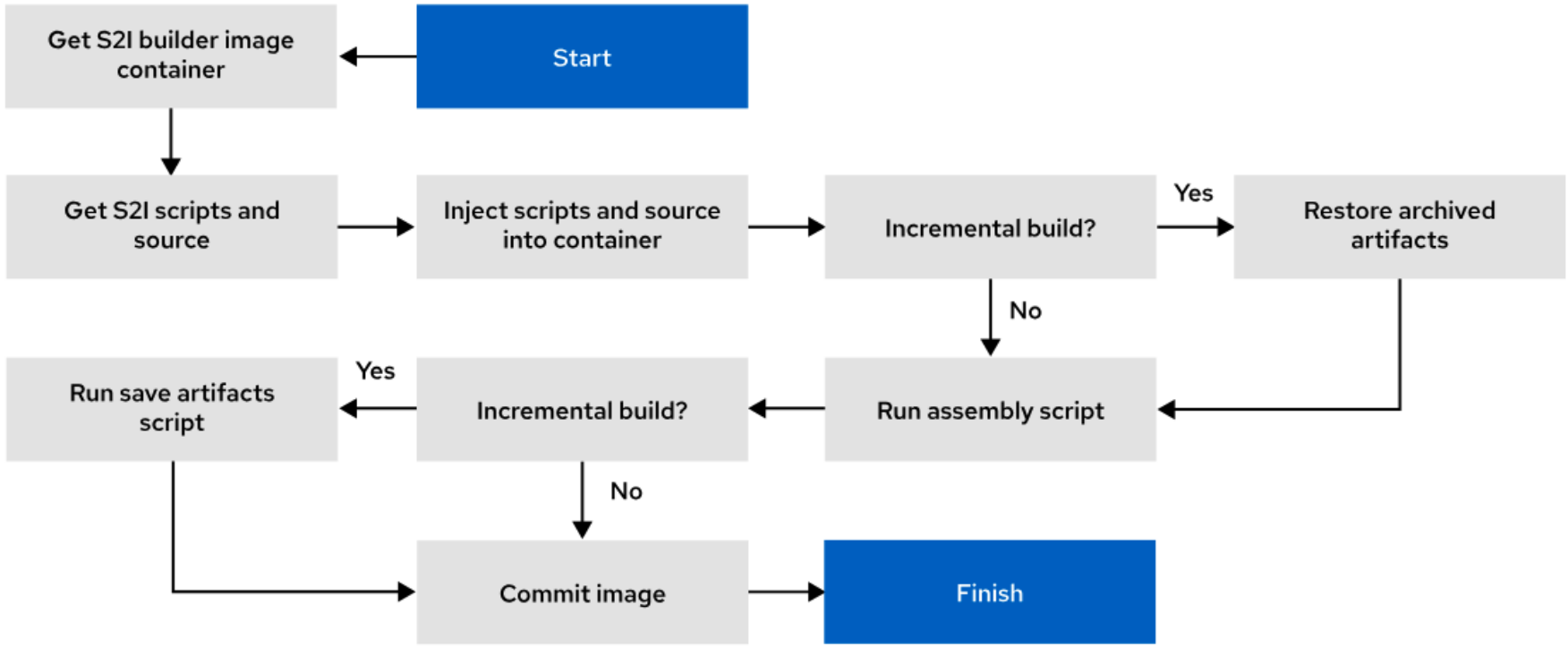
- Para forzar el uso de un determinado flujo de imágenes, se puede usar la opción -i para una aplicación PHP 7.3:

```
[user@host ~]$ oc new-app -i php:7.3 --name=php-helloworld \
 --context-dir=php-helloworld https://github.com/RedHatTraining/DO288-apps
```

- Para omitir la detección de la estrategia de compilación, puede usar la siguiente sintaxis de tilde (~):

```
[user@host ~]$ oc new-app \
php:7.3~https://github.com/RedHatTraining/DO288-apps/php-helloworld
```

FLUJO DE TRABAJO DE COMPILACIÓN S2I



CREACIÓN DE UNA IMAGEN DE COMPILADOR S2I

- El siguiente archivo de compilador Dockerfile define un compilador del servidor NGINX:

```
FROM registry.access.redhat.com/ubi8/ubi:8.0

LABEL io.k8s.description="My custom Builder" \ (1)
    io.k8s.display-name="Nginx 1.6.3" \
    io.openshift.expose-services="8080:http" \
    io.openshift.tags="builder,webserver,html,nginx" \
    io.openshift.s2i.scripts-url="image:///usr/libexec/s2i" (2)

RUN yum install -y epel-release && \ (3)
    yum install -y --nodocs nginx && \
    yum clean all

EXPOSE 8080(4)

COPY ./s2i/bin/ /usr/libexec/s2i(5)
```

1. Establezca las etiquetas que usa OpenShift para describir la imagen de compilador.
2. Establezca la ruta para los scripts obligatorios de S2I (`run`, `assemble`).
3. Instale el paquete del servidor web NGINX y limpie la memoria caché de Yum.
4. Establezca el puerto predeterminado para las aplicaciones creadas con esta imagen.
5. Copie los scripts S2I en el directorio `/usr/libexec/s2i`.

CREACIÓN DE UNA CONFIGURACIÓN DE COMPILACIÓN

- Puede usar el comando oc new-app para crear un archivo de compilación. Tenga en cuenta el siguiente comando oc new-app:

```
[user@host ~]$ oc new-app --name java-application \
--build-env BUILD_ENV=BUILD_VALUE \ (1)
--env RUNTIME_ENV=RUNTIME_VALUE \ (2)
-i redhat-openjdk18-openshift:1.8 \ (3)
--context-dir java-application \ (4)
https://git.example.com/example/java-application-repository (5)
--> Found image 11c20bc (23 months old) in image stream ...output omitted...
```

(1) Variables de entorno para los pods que realizan la compilación de la aplicación.

(2) Variables de entorno para los pods en tiempo de ejecución.

(3) Flujo de imágenes que hace referencia a la imagen del constructor S2I.

(4) La ubicación del directorio de aplicación en el repositorio de git.

(5) La ubicación del repositorio de git.

- Use la opción --strategy para usar la estrategia de compilación Docker, por ejemplo:

```
[user@host ~]$ oc new-app --name java-application \
--strategy Docker \
--context-dir java-application \
https://git.example.com/example/java-application-repository
--> Found image 11c20bc (23 months old) in image stream ...output omitted...
```

CREAR Y GESTIONAR UNA CONFIGURACIÓN DE COMPILACIÓN

CREACIÓN DE UNA CONFIGURACIÓN DE COMPILACIÓN

- Red Hat OpenShift usa el recurso de configuración de compilación para gestionar compilaciones. Puede crear y gestionar una configuración de compilación mediante la utilidad de línea de comandos (CLI) oc.

- Puede usar el comando oc new-app para crear un archivo de compilación. Tenga en cuenta el siguiente comando oc new-app:

```
[user@host ~]$ oc new-app --name java-application \
--build-env BUILD_ENV=BUILD_VALUE \ (1)
--env RUNTIME_ENV=RUNTIME_VALUE \ (2)
-i redhat-openjdk18-openshift:1.8 \ (3)
--context-dir java-application \ (4)
https://git.example.com/example/java-application-repository (5)
--> Found image 11c20bc (23 months old) in image stream ...output omitted...
```

(1) Variables de entorno para los pods que realizan la compilación de la aplicación.

(2) Variables de entorno para los pods en tiempo de ejecución.

(3) Flujo de imágenes que hace referencia a la imagen del constructor S2I.

(4) La ubicación del directorio de aplicación en el repositorio de git.

(5) La ubicación del repositorio de git.

NUEVA COMPILACIÓN

- El comando oc new-build para crear solo los recursos relacionados con la compilación:
    - La configuración de compilación
    - El origen
    - Los flujos de imágenes de destino

- Use la opción --strategy para usar la estrategia de compilación Docker, por ejemplo:

```
[user@host ~]$ oc new-app --name java-application \
--strategy Docker \
--context-dir java-application \
https://git.example.com/example/java-application-repository
--> Found image 11c20bc (23 months old) in image stream ...output omitted...
```

GESTIONAR COMPILACIONES DE LA APLICACIÓN

Puede iniciar una compilación con el comando `oc start-build`, por ejemplo:

```
[user@host ~]$ oc start-build buildconfig/app
build.build.openshift.io/app-2 started
```

Luego, puede ver los objetos de compilación:

```
[user@host ~]$ oc get builds
NAME    TYPE     FROM          STATUS     STARTED              DURATION
app-1   Source   Git@1448dc3   Complete   4 minutes ago        49s
app-2   Source   Git           Pending    About a minute ago   39s
```

Esto es útil para ver y gestionar el estado de cada compilación. Puede cancelar todas las compilaciones en ejecución mediante el comando `oc cancel-build`:

```
[user@host ~]$ oc cancel-build buildconfig/app
build.build.openshift.io/app-3 marked for cancellation, waiting to be cancelled
build.build.openshift.io/app-3 cancelled
```

Alternativamente, puede especificar las compilaciones para cancelar:

```
[user@host ~]$ oc cancel-build app-build-3
build.build.openshift.io/app-3 marked for cancellation, waiting to be cancelled
build.build.openshift.io/app-3 cancelled
```

TRIGGERING BUILDS

TIGGERS DE COMPILACIÓN

- En Red Hat OpenShift puede definir desencadenadores de compilación para iniciar nuevas compilaciones en función de determinados eventos.
- Estos son los principales desencadenadores disponibles en Red Hat OpenShift:
    - **Tiggers de cambio de imagen**
    - Tiggers de enlace web

COMPILACIONES CON TIGGER DE CAMBIO DE IMAGEN

- El comando oc new-app crea tiggers de cambio de imagen para las aplicaciones mediante estrategias de compilación Source o Docker:
    - Para la estrategia Source, la imagen principal es la imagen del compilador S2I para el lenguaje de programación de la aplicación, como openjdk-17-ubi8.
    - Para la estrategia Docker, la imagen principal es la imagen referenciada por la instrucción FROM en el Containerfile de la aplicación.
    - Para ver los cambios asociados a una configuración de compilación, usa el comando oc describe bc

```
[user@host ~]$ oc describe bc/name
...output omitted...

Webhook GitHub:
        URL:    https://api.ocp4.example.com/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<app>/webhooks/<secret>/github
Webhook Generic:
        URL:            https://api.ocp4.example.com/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<app>/webhooks/<secret>/ge
neric

...output omitted...
```

GESTION DE TIGGERS (DESENCADENADORES)

Para agregar un desencadenador de cambio de imagen a una configuración de compilación, use el comando `oc set triggers`:

```
[user@host ~]$ oc set triggers bc/name --from-image=project/image:tag
```

Una sola configuración de compilación puede incluir varios desencadenadores de cambio de imagen.

Para eliminar un desencadenador de cambio de imagen de una configuración de compilación, use el comando `oc set triggers` con la opción `--remove`:

```
[user@host ~]$ oc set triggers bc/name --from-image=project/image:tag --remove
```

Para las aplicaciones implementadas con una imagen de contenedor, puede establecer desencadenadores de la aplicación mediante el siguiente comando:

```
[user@host ~]$ oc set trigger deployment/name
```

Y puede encontrar los desencadenadores actuales para una implementación en la anotación `image.openshift.io/triggers`::

```
[user@host ~]$ oc describe deployment/name
...output omitted...

Annotations:            deployment.kubernetes.io/revision: 3
                        image.openshift.io/triggers:
[{"from":{"kind":"ImageStreamTag","name":"hello:latest"},
"fieldPath":"spec.template.spec.containers[?(@.name==\"hello\")].image"}]


...output omitted...
```

Use el comando `oc set triggers --help` a fin de ver las opciones usadas para agregar y eliminar un desencadenador de cambio de configuración.

INICIO DE NUEVAS COMPILACIONES CON TIGGERS DE ENLACE WEB

- Los tiggers de enlaces web de Red Hat OpenShift son extremos (endpoints) de API HTTP que inician nuevas compilaciones.
- Las compilaciones de Red Hat OpenShift solo pueden descargar código fuente desde un servidor de git.
- Red Hat OpenShift proporciona tipos de enlaces web especializados que admiten extremos (endpoints) API compatibles con los siguientes servicios de VCS:
    - GitLab
    - GitHub
    - BitBucket

GESTIÓN DE TIGGERS DE CAMBIO DE IMAGEN

- El comando oc new-app crea un enlace web genérico y un enlace web de git.
- Para agregar otros tipos de enlaces web a una configuración de compilación, use el comando oc set triggers.
- Por ejemplo, los siguientes comandos agregan un enlace web de GitLab a una configuración de compilación:

```
[user@host ~]$ oc set triggers bc/name --from-gitlab
```

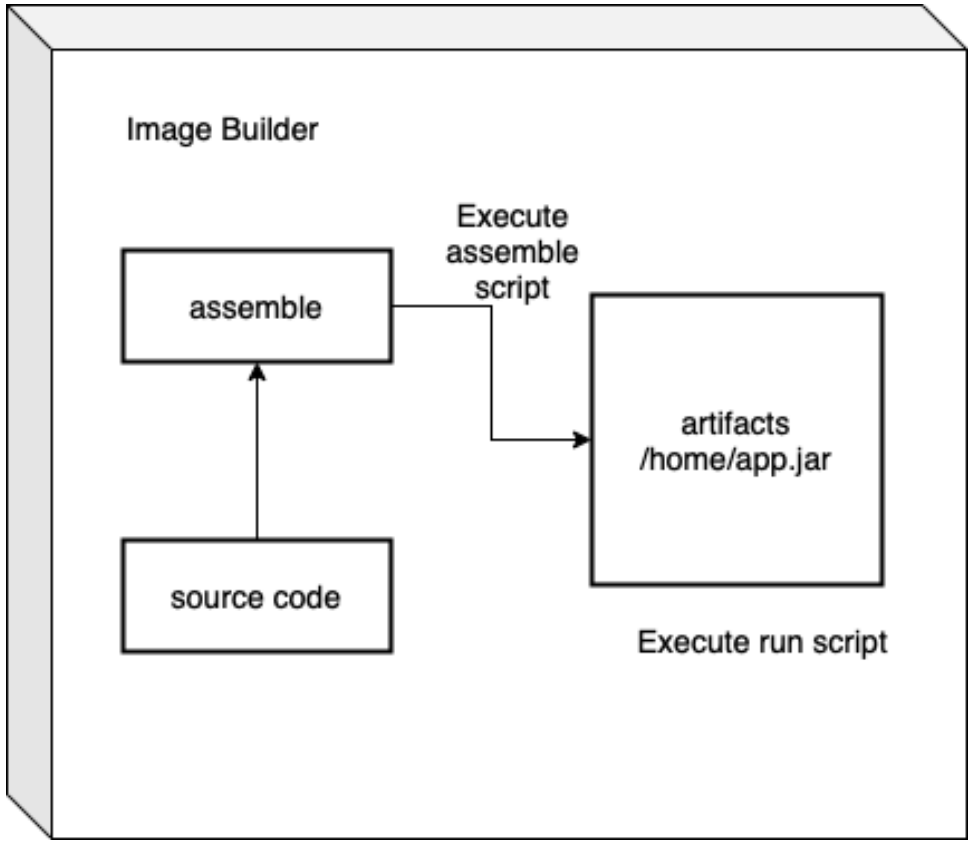
- Para eliminar un enlace web existente de una configuración de compilación, use el comando oc set triggers con la opción --remove.

```
[user@host ~]$ oc set triggers bc/name --from-gitlab --remove
```

PERSONALIZACIÓN DE UNA IMAGEN BASE S2I EXISTENTE

PERSONALIZACIÓN DE SCRIPTS DE UNA IMAGEN DE COMPILADOR S2I

- Los scripts S2I están empaquetados dentro de las imágenes del compilador S2I de manera predeterminada.
- El proceso de compilación de S2I proporciona un método para sobrescribir los scripts S2I predeterminados.
- Puede proporcionar sus propios scripts S2I en la carpeta .s2i/bin del código fuente de la aplicación



EJEMPLO DE SCRIPTS S2I

Por ejemplo, supongamos que desea personalizar los scripts S2I para la imagen de compilador S2I rhscl/php-73-rhel7 y cambiar la manera en que se compila y se ejecuta la aplicación.

- Use el comando `podman pull` para extraer la imagen de contenedor desde un registro de contenedor al sistema local. Use el comando `podman inspect` para obtener el valor de la etiqueta `io.openshift.s2i.scripts-url` y determinar la ubicación predeterminada de los scripts S2I en la imagen.

```
[user@host ~]$ podman pull \
myregistry.example.com/rhscl/php-73-rhel7
...output omitted...
Digest: sha256:...
[user@host ~]$ podman inspect \
--format='{{ index .Config.Labels "io.openshift.s2i.scripts-url"}}' \
rhscl/php-73-rhel7
image:///usr/libexec/s2i
```

- También puede usar el comando `skopeo inspect` para recuperar la misma información directamente de un registro remoto:

```
[user@host ~]$ skopeo inspect \
docker://myregistry.example.com/rhscl/php-73-rhel7 \
| grep io.openshift.s2i.scripts-url
       "io.openshift.s2i.scripts-url": "image:///usr/libexec/s2i",
```

- Cree un contenedor (wrapper) para el script `assemble` en la carpeta `.s2i/bin`:

```
#!/bin/bash
echo "Making pre-invocation changes..."

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
    echo "Making post-invocation changes..."
else
    echo "Error: assemble failed!"
fi

exit $rc
```

- De manera similar, cree un contenedor (wrapper) para el script `run` en la carpeta `.s2i/bin`:

```
#!/bin/bash
echo "Before calling run..."
exec /usr/libexec/s2i/run
```

LAB 9
GESTIÓN DE COMPILACIONES DE RED HAT OPENSHIFT