Visita guiada

Tutoriales

Manual del usuario (PDF)

- Glosario

Recursos

- Referencia de sintaxis de canal
- Referencia de Pipeline Steps
- Guía de actualización LTS

Publicaciones recientes del blog tutorial

- Presentación de tutoriales en la documentación del usuario de Jenkins
- Herramientas de desarrollo de tuberías
- Primeros pasos con Blue Ocean Dashboard

Ver todas las publicaciones del blog tutorial

⇐ Comenzar con Pipeline
⇑ Pipeline
Índice
Sucursales y solicitudes de extracción ⇒

Usando un Jenkinsfile

Tabla de contenido

Esta sección se basa en la información que se cubre en Cómo comenzar e introduce más pasos útiles, patrones comunes y muestra algunos `Jenkinsfile`ejemplos no triviales .

Crear a `Jenkinsfile`, que se registra en el control de origen [1], proporciona una serie de beneficios inmediatos:

- Revisión / iteración de código en el Pipeline
- Pista de auditoría para el Oleoducto

- Única fuente de verdad [2] para Pipeline, que puede ser vista y editada por múltiples miembros del proyecto.

Pipeline admite dos sintaxis , declarativa (introducida en Pipeline 2.5) y Scripted Pipeline. Ambos son compatibles con la construcción de conductos de entrega continua. Ambos pueden usarse para definir una interconexión en la interfaz de usuario web o con una `Jenkinsfile`, aunque generalmente se considera una mejor práctica crear `Jenkinsfile`y verificar el archivo en el repositorio de control de origen.

Creando un Jenkinsfile

Como se discutió en la sección Introducción , a `Jenkinsfile`es un archivo de texto que contiene la definición de un Gasoducto de Jenkins y está registrado en el control de fuente. Considere la siguiente canalización que implementa una canalización de entrega continua de tres etapas básicas.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

No todas las tuberías tendrán las mismas tres etapas, pero es un buen punto de partida para definirlas para la mayoría de los proyectos. Las siguientes secciones demostrarán la creación y ejecución de un Pipeline simple en una instalación de prueba de Jenkins.

> Se supone que ya hay un repositorio de control de origen configurado para el proyecto y se ha definido una interconexión en Jenkins siguiendo estas instrucciones .

Usando un editor de texto, idealmente uno que soporte el resaltado de sintaxis Groovy , cree uno nuevo `Jenkinsfile`en el directorio raíz del proyecto.

El ejemplo de Tubería declarativa anterior contiene la estructura mínima necesaria para implementar una canalización de entrega continua. La directiva de agente , que se requiere, instruye a Jenkins a asignar un ejecutor y un área de trabajo para el Pipeline. Sin una `agent`directiva, no solo el Canal Declarativo no es válido, ¡no sería capaz de hacer ningún trabajo! De forma predeterminada, la `agent`directiva garantiza que el repositorio de origen esté verificado y disponible para los pasos en las etapas posteriores.

La directiva de etapas y las directivas de pasos también son necesarias para una canalización declarativa válida, ya que le indican a Jenkins qué debe ejecutar y en qué etapa debe ejecutarse.

Para un uso más avanzado con Scripted Pipeline, el ejemplo anterior `node`es un primer paso crucial ya que asigna un ejecutor y un área de trabajo para Pipeline. En esencia, sin `node`, un Pipeline no puede hacer ningún trabajo! Desde adentro `node`, la primera orden del día será verificar el código fuente de este proyecto. Dado que

`Jenkinsfile`se está extrayendo directamente del control de fuente, Pipeline proporciona una manera rápida y fácil de acceder a la revisión correcta del código fuente

Jenkinsfile (secuencia de comandos)

```
node {
    checkout scm (1)
    /* .. snip .. */
}
```

1 El `checkout`paso registrará el código de control de origen; `scm`es una variable especial que indica el `checkout`paso para clonar la revisión específica que activó esta ejecución de Pipeline.

Construir

Para muchos proyectos, el comienzo del "trabajo" en el Pipeline sería la etapa de "construcción". Típicamente, esta etapa de Pipeline será donde se ensamble, compile o empaque el código fuente. El `Jenkinsfile`es **no** un reemplazo para una herramienta de construcción existentes, tales como GNU / Hacer, Maven, Gradle, etc, sino que puede ser visto como una capa de pegamento para unir las múltiples fases del ciclo de desarrollo de un proyecto (construir, probar, implementar, etc.) juntos.

Jenkins tiene una serie de complementos para invocar prácticamente cualquier herramienta de compilación de uso general, pero este ejemplo simplemente invocará `make`desde un paso de shell (`sh`). El `sh`paso supone que el sistema está basado en Unix / Linux, para los sistemas basados en Windows `bat`podría utilizarse en su lugar.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'make' (1)
                archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true (2)
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

1 El `sh`paso invoca el `make`comando y solo continuará si el comando devuelve un código de salida cero. Cualquier código de salida distinto de cero fallará en Pipeline.

2 `archiveArtifacts`captura los archivos creados que coinciden con el patrón de inclusión (`**/target/*.jar`) y los guarda en el maestro de Jenkins para su posterior recuperación.

Archivar artefactos no es un sustituto del uso de repositorios de artefactos externos, como Artifactory o Nexus, y se debe considerar solo para informes básicos y archivo de archivos.

Prueba

Ejecutar pruebas automatizadas es un componente crucial de cualquier proceso de entrega continua exitoso. Como tal, Jenkins tiene una serie de instalaciones de prueba de grabación, informes y visualización proporcionadas por una serie de complementos . En un nivel fundamental, cuando hay fallas en las pruebas, es útil que Jenkins registre las fallas para informar y visualizar en la interfaz de usuario web. El siguiente ejemplo utiliza el `junit`paso, proporcionado por el complemento JUnit .

En el ejemplo siguiente, si las pruebas fallan, la tubería se marca como "inestable", como se indica mediante una bola amarilla en la interfaz de usuario web. Con base en los informes de prueba registrados, Jenkins también puede proporcionar análisis y visualización de tendencias históricas.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any

    stages {
        stage('Test') {
            steps {
                /* `make check` returns non-zero on test failures,
                * using `true` to allow the Pipeline to continue nonetheless
                */
                sh 'make check || true' (1)
                junit '**/target/*.xml' (2)
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

1 El uso de un shell condicional en línea (`sh 'make || true'`) garantiza que el `sh`paso siempre vea un código de salida cero, dando al `junit`paso la oportunidad de capturar y procesar los informes de prueba. Los enfoques alternativos a esto se tratan con más detalle en la sección de falla de manejo a continuación.

2 `junit`captura y asocia los archivos de JUnit XML que coinciden con el patrón de inclusión (`**/target/*.xml`).

Desplegar

La implementación puede implicar una variedad de pasos, según los requisitos del proyecto o de la organización, y puede ser cualquier cosa, desde la publicación de artefactos construidos hasta un servidor Artifactory, hasta el envío de código a un sistema de producción.

En esta etapa del ejemplo Pipeline, las etapas "Build" y "Test" se han ejecutado con éxito. En esencia, la etapa "Desplegar" solo se ejecutará suponiendo que las etapas anteriores se completaron con éxito, de lo contrario, el Ducto habría salido temprano.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any

    stages {
        stage('Deploy') {
            when {
              expression {
                currentBuild.result == null || currentBuild.result == 'SUCCESS' (1)
              }
            }
            steps {
                sh 'make publish'
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

1 El acceso a la `currentBuild.result`variable permite que Pipeline determine si hubo fallas en la prueba. En cuyo caso, el valor sería `UNSTABLE`.

Suponiendo que todo se haya ejecutado correctamente en el ejemplo de Jenkins Pipeline, cada ejecución exitosa de Pipeline tendrá archivados los artefactos de construcción asociados, los resultados de las pruebas informados y la salida de la consola completa, todo en Jenkins.

Una canalización con scripts puede incluir pruebas condicionales (como se muestra arriba), bucles, bloques try / catch / finally e incluso funciones. La siguiente sección cubrirá esta sintaxis avanzada de Pipeline con scripts con más detalle.

Trabajando con su Jenkinsfile

Las siguientes secciones proporcionan detalles sobre el manejo:

- sintaxis específica de Pipeline en su `Jenkinsfiley`
- características y funcionalidad de la sintaxis de Pipeline que son esenciales en la construcción de su aplicación o proyecto Pipeline.

Interpolación de cadenas

Jenkins Pipeline usa reglas idénticas a Groovy para la interpolación de cadenas. El soporte de interpolación String de Groovy puede ser confuso para muchos recién llegados al idioma. Mientras que Groovy admite declarar una cadena con comillas simples o comillas dobles, por ejemplo:

```
def singlyQuoted = 'Hello'
def doublyQuoted = "World"
```

Solo la última cadena admitirá la `$`interpolación de cadenas basada en signo de dólar ( ), por ejemplo:

```
def username = 'Jenkins'
echo 'Hello Mr. ${username}'
echo "I said, Hello Mr. ${username}"
```

Daría lugar a:

```
Hello Mr. ${username}
I said, Hello Mr. Jenkins
```

Entender cómo usar la interpolación de cadenas es vital para usar algunas de las características más avanzadas de Pipeline.

Usando variables de entorno

Jenkins Pipeline expone variables de entorno a través de la variable global `env`, que está disponible desde cualquier lugar dentro de a `Jenkinsfile`. La lista completa de las variables de entorno accesibles desde Jenkins Pipeline está documentada en localhost: 8080 / pipeline-syntax / globals # env , suponiendo que se ejecuta un master de Jenkins `localhost:8080`, e incluye:

BUILD_ID

> La ID de compilación actual, idéntica a BUILD_NUMBER para compilaciones creadas en Jenkins versiones 1.597+

NOMBRE DEL TRABAJO

> Nombre del proyecto de esta compilación, como "foo" o "foo / bar".

JENKINS_URL

> URL completa de Jenkins, como example.com:port/jenkins/ (NOTA: solo está disponible si la URL de Jenkins está configurada en "Configuración del sistema")

Se puede lograr la referencia o el uso de estas variables de entorno, como acceder a cualquier clave en un mapa de Groovy , por ejemplo:

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
```

```
                echo "Running ${env.BUILD_ID} on ${env.JENKINS_URL}"
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

Establecer variables de entorno

La configuración de una variable de entorno dentro de un Pipeline de Jenkins se realiza de forma diferente dependiendo de si se utiliza el Pipeline Declarativo o Scripted.

Declarative Pipeline admite una directiva de entorno , mientras que los usuarios de Scripted Pipeline deben usar el `withEnv`paso.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any
    environment { (1)
        CC = 'clang'
    }
    stages {
        stage('Example') {
            environment { (2)
                DEBUG_FLAGS = '-g'
            }
            steps {
                sh 'printenv'
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

1 Una `environment`directiva utilizada en el `pipeline`bloque de nivel superior se aplicará a todos los pasos dentro de Pipeline.

2 Una `environment`directiva definida dentro de un `stage`solo aplicará las variables de entorno dadas a los pasos dentro de `stage`.

Manejo de credenciales

Las credenciales configuradas en Jenkins pueden manejarse en tuberías para uso inmediato. Obtenga más información sobre cómo usar las credenciales en Jenkins en la página Usar credenciales .

Para texto secreto, nombres de usuario y contraseñas, y archivos secretos

La sintaxis de Pipeline descompensada de Jenkins tiene el `credentials()`método de ayuda (utilizado dentro de la `environment`directiva) que admite texto secreto , nombre de usuario y contraseña , así como credenciales de archivos secretas . Si desea manejar otros tipos de credenciales, consulte la sección Para otros tipos de credenciales (a continuación).

Texto secreto

El siguiente código de Pipeline muestra un ejemplo de cómo crear un Pipeline usando variables de entorno para credenciales de texto secreto.

En este ejemplo, se asignan dos credenciales de texto secreto a variables de entorno separadas para acceder a Amazon Web Services (AWS). Estas credenciales se habrían configurado en Jenkins con sus respectivas credenciales ID
`jenkins-aws-secret-key-id`y `jenkins-aws-secret-access-key`.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent {
        // Define agent details here
    }
    environment {
        AWS_ACCESS_KEY_ID     = credentials('jenkins-aws-secret-key-id')
        AWS_SECRET_ACCESS_KEY = credentials('jenkins-aws-secret-access-key')
    }
    stages {
        stage('Example stage 1') {
            steps {
                // (1)
            }
        }
        stage('Example stage 2') {
            steps {
                // (2)
            }
        }
    }
}
```

Puede hacer referencia a las dos variables de entorno de credenciales (definidas en la `environment`directiva de este Pipeline), dentro de los pasos de esta etapa utilizando la sintaxis `$AWS_ACCESS_KEY_ID`y `$AWS_SECRET_ACCESS_KEY`. Por ejemplo, aquí puede autenticarse en AWS utilizando las credenciales de texto secreto asignadas a estas variables de credenciales.

1 Para mantener la seguridad y el anonimato de estas credenciales, si intenta recuperar el valor de estas variables de credencial desde Pipeline (p `echo $AWS_SECRET_ACCESS_KEY`. Ej.), Jenkins solo devuelve el valor " ****" para evitar que se escriba información secreta en la salida de la consola y cualquier registro . Cualquier información confidencial en los ID de credenciales (como los nombres de usuario) también se devuelve como " ****" en el resultado de la ejecución de Pipeline.

2 En este ejemplo de Pipeline, las credenciales asignadas a las dos `AWS_...` variables de entorno tienen un alcance global para todo el Pipeline, por lo que estas variables de credenciales también se podrían usar en los pasos de esta etapa. Sin embargo, si la `environment`directiva en este Pipeline se movió a una etapa específica (como es el caso en el ejemplo Pipeline de nombres de usuario y contraseñas a continuación), estas `AWS_...`variables de entorno solo se ajustarán a los pasos en esa etapa.

Nombres de usuario y contraseñas

Los siguientes fragmentos de código de Pipeline muestran un ejemplo de cómo crear un Pipeline utilizando variables de entorno para las credenciales de nombre de usuario y contraseña.

En este ejemplo, las credenciales de nombre de usuario y contraseña se asignan a variables de entorno para acceder a un repositorio de Bitbucket en una cuenta o equipo común para su organización; estas credenciales se habrían configurado en Jenkins con la identificación de credencial `jenkins-bitbucket-common-creds`.

Al configurar la variable de entorno de credenciales en la `environment`directiva:

```
environment {
    BITBUCKET_COMMON_CREDS = credentials('jenkins-bitbucket-common-creds')
}
```

esto en realidad establece las siguientes tres variables de entorno:

- `BITBUCKET_COMMON_CREDS`- contiene un nombre de usuario y una contraseña separados por dos puntos en el formato `username:password`.
- `BITBUCKET_COMMON_CREDS_USR` - una variable adicional que contiene solo el componente de nombre de usuario.
- `BITBUCKET_COMMON_CREDS_PSW` - una variable adicional que contiene solo el componente de contraseña.

Por convención, los nombres de variables para las variables de entorno se suelen especificar en mayúsculas, con palabras individuales separadas por guiones bajos. Sin embargo, puede especificar cualquier nombre de variable legítimo utilizando caracteres en minúscula. Tenga en cuenta que las variables de entorno adicionales creadas por el `credentials()` método (arriba) siempre se anexarán con `_USR`y `_PSW`(es decir, en el formato de un guión bajo seguido de tres letras en mayúscula).

El siguiente fragmento de código muestra el ejemplo Pipeline en su totalidad:

Jenkinsfile (canal declarativo)

```
pipeline {
    agent {
        // Define agent details here
    }
    stages {
        stage('Example stage 1') {
            environment {
                BITBUCKET_COMMON_CREDS = credentials('jenkins-bitbucket-common-creds')
            }
            steps {
                // (1)
            }
        }
        stage('Example stage 2') {
            steps {
                // (2)
            }
        }
    }
}
```

Las siguientes variables de entorno de credenciales (definidas en la `environment`directiva de Pipeline) están disponibles dentro de los pasos de esta etapa y se pueden referenciar utilizando la sintaxis:

- `$BITBUCKET_COMMON_CREDS`
- `$BITBUCKET_COMMON_CREDS_USR`

1

- `$BITBUCKET_COMMON_CREDS_PSW`

Por ejemplo, aquí puede autenticarse en Bitbucket con el nombre de usuario y la contraseña asignados a estas variables de credencial.
Para mantener la seguridad y el anonimato de estas credenciales, si intenta recuperar el valor de estas variables de credencial desde Pipeline, el mismo comportamiento descrito en el ejemplo de texto secreto anterior también se aplica a estos tipos de variable de credenciales de nombre de usuario y contraseña.

2

En este ejemplo de Pipeline, las credenciales asignadas a las tres `COMMON_BITBUCKET_CREDS...`variables de entorno tienen un ámbito exclusivo `Example stage 1`, por lo que estas variables de credenciales no están disponibles para su uso en `Example stage 2`los pasos de esta etapa. Sin embargo, si la `environment`directiva en este Pipeline se movió inmediatamente dentro del `pipeline`bloque (como es el caso en el ejemplo de Pipeline de texto secreto anterior), entonces estas `COMMON_BITBUCKET_CREDS...`variables de entorno serían consideradas globalmente y podrían usarse en los pasos de cualquier etapa.

Archivos secretos

En lo que respecta a Pipelines, los archivos secretos se manejan exactamente de la misma manera que el texto secreto (arriba).

Esencialmente, la única diferencia entre el texto secreto y las credenciales de los archivos secretos es que para el texto secreto, la credencial se ingresa directamente en Jenkins, mientras que para un archivo secreto, la credencial se almacena originalmente en un archivo que luego se carga en Jenkins.

A diferencia del texto secreto, los archivos secretos satisfacen credenciales que son:

- demasiado difícil de manejar para ingresar directamente a Jenkins, y / o
- en formato binario, como un archivo GPG.

Para otros tipos de credenciales

Si necesita establecer credenciales en Pipeline para algo que no sea texto secreto, nombres de usuario y contraseñas, o archivos secretos (arriba), es decir, claves SSH o certificados, utilice la función Jenkins ' **Snippet Generator** , a la que puede acceder a través de la IU clásica de Jenkins .

Para acceder al **Generador de Fragmentos** para su proyecto / artículo del Ducto:

1. Desde la página de inicio de Jenkins (es decir, el panel de la interfaz de usuario clásica de Jenkins), haga clic en el nombre de su proyecto / elemento de Pipeline.
2. A la izquierda, haga clic en **Sintaxis de Pipeline** y asegúrese de que el enlace del **generador de fragmentos** esté en negrita en la esquina superior izquierda. (De lo contrario, haga clic en su enlace).
3. Desde el campo **Paso de muestra** , elija **conCredentials: vincular credenciales a variables** .
4. En **Vinculaciones** , haga clic en **Agregar** y elija en el menú desplegable:
    - **Clave privada de usuario SSH** : para manejar las credenciales de par de claves públicas / privadas de SSH , desde las cuales puede especificar:
        - **Variable de archivo clave** : el nombre de la variable de entorno que se vinculará a estas credenciales. De hecho, Jenkins asigna esta variable temporal a la ubicación segura del archivo de clave privada requerido en el proceso de autenticación de par de clave pública / privada de SSH.
        - **Variable de** contraseña (*opcional*): el nombre de la variable de entorno que se vinculará a la frase de contraseña asociada con el par de clave pública / privada SSH.
        - **Variable de** nombre de **usuario** (*opcional*): el nombre de la variable de entorno que se vinculará al nombre de usuario asociado con el par de claves públicas / privadas de SSH.
        - **Credenciales** : elija las credenciales de clave pública / privada SSH almacenadas en Jenkins. El valor de este campo es la ID de credencial, que Jenkins escribe en el fragmento generado.
    - **Certificado** : para manejar certificados PKCS # 12 , desde los cuales puede especificar:
        - **Variable de almacén de claves** : el nombre de la variable de entorno que se vinculará a estas credenciales. De hecho, Jenkins asigna esta variable temporal a la ubicación segura del almacén de claves del certificado requerido en el proceso de autenticación del certificado.
        - **Variable de contraseña** (*opcional*): el nombre de la variable de entorno que se vinculará a la contraseña asociada con el certificado.
        - **Variable alias** (*opcional*): el nombre de la variable de entorno que se vinculará con el alias único asociado con el certificado.
        - **Credenciales** : elija las credenciales de certificado almacenadas en Jenkins. El valor de este campo es la ID de credencial, que Jenkins escribe en el fragmento generado.
    - **Certificado de cliente de Docker** : para gestionar la autenticación de certificado de host Docker.
5. Haga clic en **Generar secuencia de comandos de canalización** y Jenkins genera un `withCredentials( ... ) { ... }`fragmento de paso de canalización para las credenciales que ha especificado, que luego puede copiar y pegar en su código de canalización declarativo o por **secuencias de comandos** .
Notas:

- Los campos **Credenciales** (arriba) muestran los nombres de las credenciales configuradas en Jenkins. Sin embargo, estos valores se convierten en ID de credenciales después de hacer clic en **Generar secuencia de comandos de canalización** .
- Para combinar más de una credencial en un solo `withCredentials( ... ) { ... }`paso de Pipeline, vea Combinar credenciales en un paso (abajo) para más detalles.

Ejemplo de clave privada de usuario SSH

```
withCredentials(bindings: [sshUserPrivateKey(credentialsId: 'jenkins-ssh-key-for-abc', \
                                             keyFileVariable: 'SSH_KEY_FOR_ABC', \
                                             passphraseVariable: '', \
                                             usernameVariable: '')]) {
  // some block
}
```

Las definiciones opcionales `passphraseVariable`y `usernameVariable`se pueden eliminar en su código final de Pipeline.

Ejemplo de certificado

```
withCredentials(bindings: [certificate(aliasVariable: '', \
                                       credentialsId: 'jenkins-certificate-for-xyz', \
                                       keystoreVariable: 'CERTIFICATE_FOR_XYZ', \
                                       passwordVariable: 'XYZ-CERTIFICATE-PASSWORD')]) {
  // some block
}
```

Las definiciones opcionales `aliasVariable`y `passwordVariable`variables se pueden eliminar en su código final de Pipeline.

El siguiente fragmento de código muestra un ejemplo Pipeline en su totalidad, que implementa la **clave privada de usuario SSH** y los fragmentos de **certificados** anteriores:

Jenkinsfile (canal declarativo)

```
pipeline {
    agent {
        // define agent details
    }
    stages {
        stage('Example stage 1') {
            steps {
                withCredentials(bindings: [sshUserPrivateKey(credentialsId: 'jenkins-ssh-key-for-abc', \
                                                             keyFileVariable: 'SSH_KEY_FOR_ABC')]) {
                  // (1)
                }
                withCredentials(bindings: [certificate(credentialsId: 'jenkins-certificate-for-xyz', \
                                                       keystoreVariable: 'CERTIFICATE_FOR_XYZ', \
                                                       passwordVariable: 'XYZ-CERTIFICATE-PASSWORD')]) {
                  // (2)
                }
            }
        }
        stage('Example stage 2') {
            steps {
                // (3)
            }
        }
    }
}
```

1. Dentro de este paso, puede hacer referencia a la variable de entorno de credenciales con la sintaxis `$SSH_KEY_FOR_ABC`. Por ejemplo, aquí puede autenticarse en la aplicación ABC con sus credenciales de pares de clave pública / privada SSH configuradas, a las que se asigna el archivo de **clave privada de usuario SSH**`$SSH_KEY_FOR_ABC` .
2. Dentro de este paso, puede hacer referencia a la variable de entorno de credenciales con la sintaxis

`$CERTIFICATE_FOR_XYZ`y `$XYZ-CERTIFICATE-PASSWORD`. Por ejemplo, aquí puede autenticarse en la aplicación XYZ con sus credenciales de certificado configuradas, cuyo archivo y contraseña de almacén de **certificados** se asignan a las variables `$CERTIFICATE_FOR_XYZ` y `$XYZ-CERTIFICATE-PASSWORD`, respectivamente.

3 En este ejemplo de Pipeline, las credenciales asignadas a `$SSH_KEY_FOR_ABC`, `$CERTIFICATE_FOR_XYZ`y `$XYZ-CERTIFICATE-PASSWORD`las variables de entorno tienen un alcance solo dentro de sus respectivos `withCredentials( ... ) { ... }`pasos, por lo que estas variables de credenciales no están disponibles para su uso en `Example stage 2`los pasos de esta etapa.

Para mantener la seguridad y el anonimato de estas credenciales, si intenta recuperar el valor de estas variables de credenciales desde estos `withCredentials( ... ) { ... }`pasos, el mismo comportamiento descrito en el ejemplo de texto secreto (arriba) se aplica a estas credenciales de par de claves públicas / privadas SSH y certificado tipos de variables también.

- Al usar las **opciones** del campo **Paso de muestra con Credenciales: vincular credenciales a variables** en el **Generador de fragmentos** , solo las credenciales a las que tenga acceso su proyecto / elemento de Ducto actual se pueden seleccionar desde cualquier lista del campo **Credenciales** . Si bien puede escribir manualmente un `withCredentials( ... ) { ... }`paso para su Pipeline (como los ejemplos anteriores), se recomienda usar el **generador de fragmentos** para evitar especificar credenciales que están fuera del alcance de este proyecto / elemento Pipeline, que cuando se ejecuta, hará que el paso falle.
- También puede usar el **generador de fragmentos** para generar `withCredentials( ... ) { ... }`pasos para manejar el texto secreto, nombres de usuario y contraseñas y archivos secretos. Sin embargo, si solo necesita manejar estos tipos de credenciales, se recomienda que use el procedimiento relevante descrito en la sección anterior para una mejor legibilidad del código de Pipeline.

Combinar credenciales en un solo paso

Con el **generador de fragmentos** , puede hacer que varias credenciales estén disponibles en un solo `withCredentials( ... ) { ... }`paso haciendo lo siguiente:

1. Desde la página de inicio de Jenkins (es decir, el panel de la interfaz de usuario clásica de Jenkins), haga clic en el nombre de su proyecto / elemento de Pipeline.
2. A la izquierda, haga clic en **Sintaxis de Pipeline** y asegúrese de que el enlace del **generador de fragmentos** esté en negrita en la esquina superior izquierda. (De lo contrario, haga clic en su enlace).
3. Desde el campo **Paso de muestra** , elija **conCredentials: vincular credenciales a variables** .
4. Haga clic en **Agregar** debajo de **Vinculaciones** .
5. Elija el tipo de credencial para agregar al `withCredentials( ... ) { ... }`paso de la lista desplegable.
6. Especifique los detalles de **enlaces de** credenciales . Lea más sobre esto en el procedimiento bajo Para otros tipos de credenciales (arriba).
7. Repita desde "Haga clic en **Agregar** ..." (arriba) para cada (conjunto de) credenciales / s para agregar al `withCredentials( ... ) { ... }`paso.
8. Haga clic en **Generar secuencia** de **comandos de tubería** para generar el último `withCredentials( ... ) { ... }`fragmento de paso.

Manejo de parámetros

Declarative Pipeline admite parámetros listos para usar, lo que permite a Pipeline aceptar los parámetros especificados por el usuario en tiempo de ejecución a través de la directiva de parámetros . La configuración de parámetros con Scripted Pipeline se realiza con el `properties`paso, que se puede encontrar en el generador de fragmentos.

Si configuró su canalización para aceptar parámetros utilizando la opción **Construir con parámetros** , se puede acceder a esos parámetros como miembros de la `params` variable.

Suponiendo que un parámetro de cadena llamado "Saludo" se haya configurado en el `Jenkinsfile`, puede acceder a ese parámetro a través de `${params.Greeting}`:

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any
    parameters {
        string(name: 'Greeting', defaultValue: 'Hello', description: 'How should I greet the world?')
    }
    stages {
        stage('Example') {
            steps {
                echo "${params.Greeting} World!"
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

Manejo de fallas

Declarative Pipeline apoya robusto el control de fallos por defecto a través de su sección de poste que permite declarar un número de diferentes "condiciones de enviar", tales como: `always`, `unstable`, `success`, `failure`, y `changed`. La sección Sintaxis de canalización proporciona más detalles sobre cómo usar las diversas condiciones de publicación.

Jenkinsfile (canal declarativo)

```
pipeline {
    agent any
    stages {
        stage('Test') {
            steps {
                sh 'make check'
            }
        }
    }
    post {
        always {
            junit '**/target/*.xml'
        }
        failure {
            mail to: team@example.com, subject: 'The Pipeline failed :('
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

Sin embargo, Scripted Pipeline se basa en la semántica `try`/ `catch`/ `finally`semántica incorporada de Groovy para gestionar fallas durante la ejecución de Pipeline.

En el ejemplo de prueba anterior, el `sh`paso se modificó para que nunca se devuelva un código de salida distinto de cero (`sh 'make check || true'`). Este enfoque, si bien es válido, significa que las siguientes etapas deben verificar `currentBuild.result`si se ha producido un error en la prueba o no.

Una forma alternativa de manejar esto, que preserva el comportamiento de salida temprana de fallas en Pipeline, mientras se da `junit`la oportunidad de capturar informes de prueba, es usar una serie de `try`/ `finally`bloques:

Usando múltiples agentes

En todos los ejemplos anteriores, solo se ha utilizado un único agente. Esto significa que Jenkins asignará un ejecutor donde sea que esté disponible, independientemente de cómo esté etiquetado o configurado. No solo se puede anular este comportamiento, sino que Pipeline permite utilizar varios agentes en el entorno de Jenkins desde dentro del *mismo* `Jenkinsfile` , lo que puede ser útil para casos de uso más avanzados, como ejecutar compilaciones / pruebas en múltiples plataformas.

En el ejemplo siguiente, la etapa "Generar" se realizará en un agente y los resultados generados se reutilizarán en dos agentes posteriores, etiquetados como "linux" y "windows" respectivamente, durante la etapa "Prueba".

Jenkinsfile (canal declarativo)

```
pipeline {
    agent none
    stages {
        stage('Build') {
            agent any
            steps {
                checkout scm
                sh 'make'
                stash includes: '**/target/*.jar', name: 'app' (1)
            }
        }
        stage('Test on Linux') {
            agent { (2)
                label 'linux'
            }
            steps {
                unstash 'app' (3)
                sh 'make check'
            }
            post {
                always {
                    junit '**/target/*.xml'
                }
            }
        }
        stage('Test on Windows') {
            agent {
                label 'windows'
            }
            steps {
                unstash 'app'
                bat 'make check' (4)
            }
            post {
                always {
                    junit '**/target/*.xml'
                }
            }
        }
    }
}
```

Alternar secuencia de comandos con script *(avanzado)*

1 El `stash`paso permite capturar archivos que coinciden con un patrón de inclusión (`**/target/*.jar`) para su reutilización dentro de la *misma* canalización. Una vez que Pipeline ha completado su ejecución, los archivos escondidos se eliminan del maestro de Jenkins.

2 El parámetro en `agent`/ `node`permite cualquier expresión de etiqueta de Jenkins válida. Consulte la sección Sintaxis de Pipeline para más detalles.

3 `unstash` recuperará el "alijo" nombrado del maestro de Jenkins en el espacio de trabajo actual del Oleoducto.

4 El `bat`script permite ejecutar scripts por lotes en plataformas basadas en Windows.

Argumentos de paso opcionales

Pipeline sigue la convención del lenguaje Groovy de permitir omitir paréntesis alrededor de los argumentos del método.

Muchos pasos de Pipeline también usan la sintaxis de parámetros nombrados como una abreviatura para crear un Mapa en Groovy, que usa la sintaxis `[key1: value1, key2: value2]`. Hacer afirmaciones como las siguientes funcionalmente equivalentes:

```
git url: 'git://example.com/amazing-project.git', branch: 'master'
git([url: 'git://example.com/amazing-project.git', branch: 'master'])
```

Para mayor comodidad, al llamar a los pasos que toman solo un parámetro (o solo un parámetro obligatorio), el nombre del parámetro puede omitirse, por ejemplo:

```
sh 'echo hello' /* short form  */
sh([script: 'echo hello'])  /* long form */
```

Tubería avanzada de scripts

Scripted Pipeline es un lenguaje específico de dominio [3] basado en Groovy, la mayoría de la sintaxis de Groovy se puede usar en Scripted Pipeline sin modificaciones.

Ejecución en paralelo

El ejemplo en la sección anterior ejecuta pruebas en dos plataformas diferentes en una serie lineal. En la práctica, si la `make check` ejecución tarda 30 minutos en completarse, la etapa "Prueba" ahora demorará 60 minutos en completarse.

Afortunadamente, Pipeline tiene una funcionalidad incorporada para ejecutar porciones de Scripted Pipeline en paralelo, implementadas en el `parallel`paso apropiado .

Refactorizando el ejemplo anterior para usar el `parallel`paso:

Jenkinsfile (secuencia de comandos)

```
stage('Build') {
    /* .. snip .. */
}

stage('Test') {
    parallel linux: {
        node('linux') {
            checkout scm
            try {
                unstash 'app'
                sh 'make check'
            }
            finally {
                junit '**/target/*.xml'
            }
        }
    },
    windows: {
        node('windows') {
            /* .. snip .. */
        }
    }
}
```

En lugar de ejecutar las pruebas en los nodos etiquetados "linux" y "windows" en serie, ahora se ejecutarán en paralelo suponiendo que exista la capacidad requerida en el entorno de Jenkins.

---

1 . en.wikipedia.org/wiki/Source_control_management
2 . en.wikipedia.org/wiki/Single_Source_of_Truth
3 . en.wikipedia.org/wiki/Domain-specific_language

⇐ Comenzar con Pipeline
⇑ Pipeline

Índice
Sucursales y solicitudes de extracción ⇒

¿Fue útil esta página

Please submit your feedback about this page through this quick form.

Alternatively, if you don't wish to complete the quick form, you can simply indicate if you found this page helpful?

Yes No

Type the answer to 7 plus 10 before clicking "Submit" below.

Submit

See existing feedback here.

Mejora esta página | Historial de la página

Recursos

- Eventos
- Documentación
- Blog

Soluciones

- Androide
- C / C ++
- Estibador
- Incrustado
- GitHub
- Java
- PHP
- Entrega continua
- Pitón
- Rubí

Proyecto

- Seguimiento de problemas
- Wiki
- GitHub
- Jenkins en Jenkins

Comunidad

- Lista de correo de usuarios
- Lista de correo de desarrolladores
- Gorjeo
- Reddit
- Mercancías