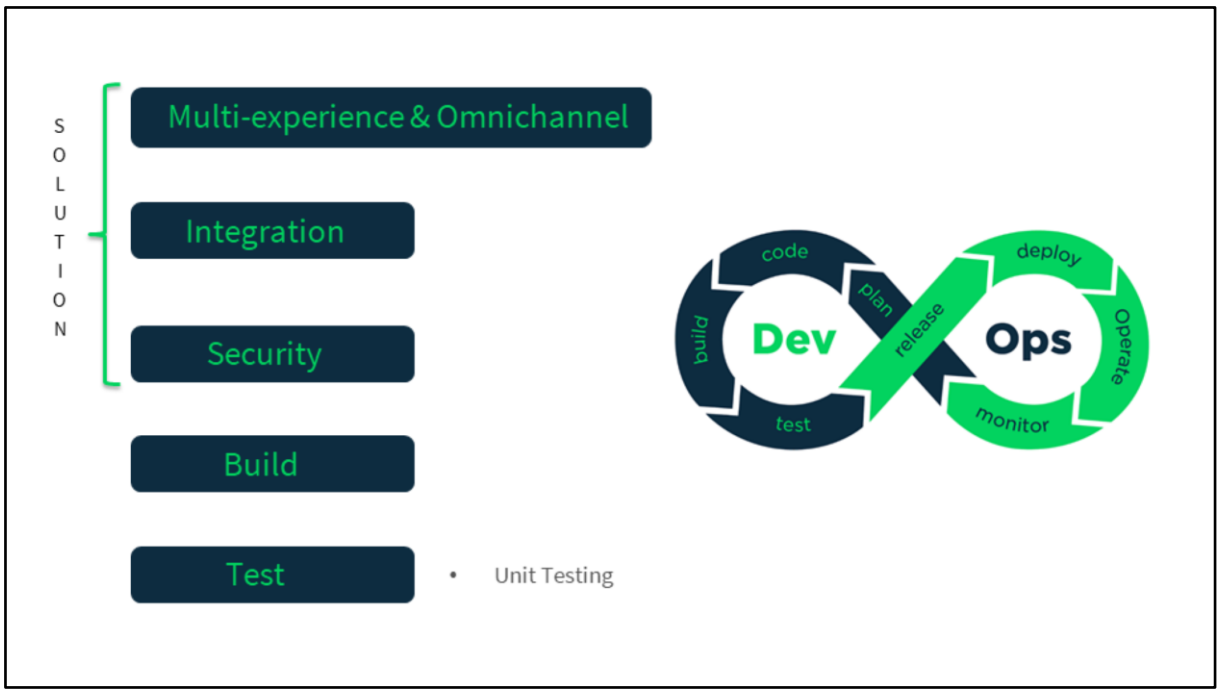
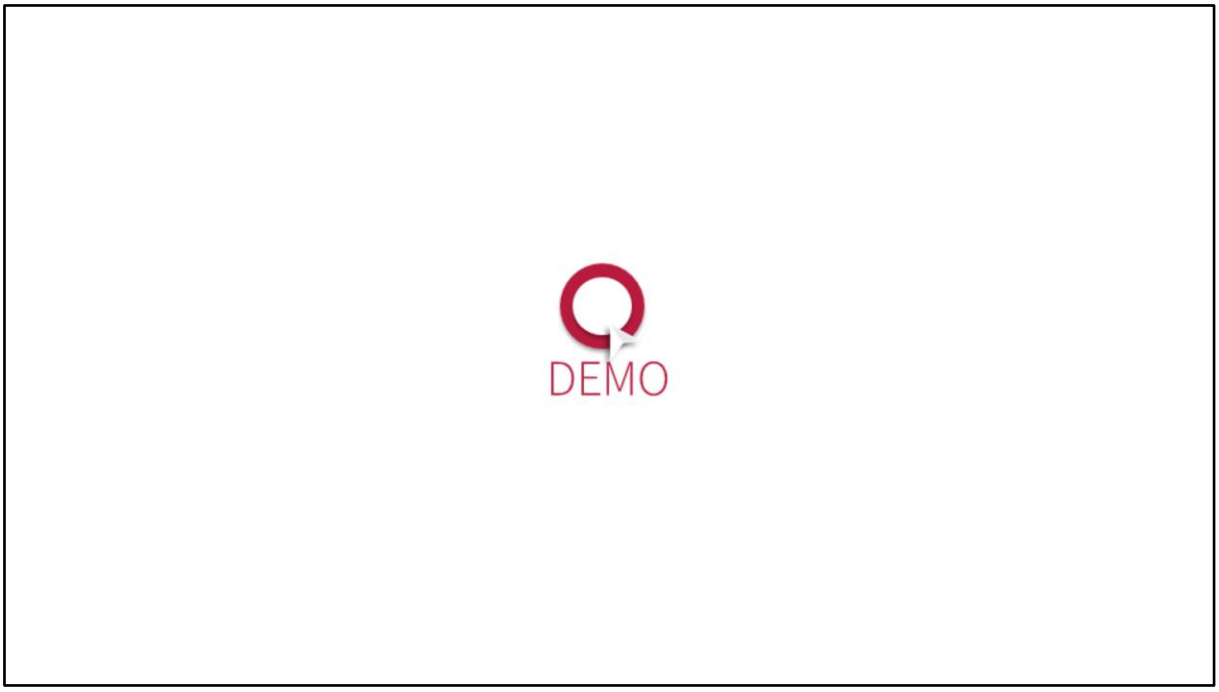

GeneXus™
The power of doing



Hasta aquí hemos atravesado las primeras partes del diagrama de DevOps, los aspectos que hacen a la aplicación que estamos desarrollando y cómo hacerlo con GeneXus.

Sobre Build ya hablamos en la introducción, así que no agregaremos nada aquí, vamos a ver ahora que es lo nuevo en Testing.

A nivel de Testing, tenemos la posibilidad de crear Unit Test integrados desde el IDE, con los Unit Test vamos a poder testear tanto procedimientos como Datas Providers, vamos a ver la idea con un ejemplo básico…



Tenemos éste procedimiento que nos permite dar de alta una queja, cuando damos botón derecho sobre el mismo, tenemos una nueva opción que es crear Unit Test, esto nos va a crear la estructura básica de un Test, lo que tenemos aquí es un objeto nuevo de tipo Unit Test que se programa de la misma forma que programamos un procedimiento y viene pre-cargado con una estructura que nos simplifica el trabajo.

En la sesión Act, vemos que se invoca el procedimiento que queremos testear con todos sus parámetros cargados a partir de un TestCaseData, (que ahora vamos a ver de donde sale) y Response, que es el valor de salida del procedimiento que estamos testeando.

En la sesión Assert, tenemos la validación del resultado obtenido, que sería el de la variable Response contra un valor esperado que también se define en el TestCaseData,
TestCaseData es una estructura que contiene los datos para cada caso de prueba y al crear el Test también se crea automáticamente un Data Provider que nos facilita la carga de nuestros datos de prueba.

Vamos a comenzar haciendo un Test para lo que sería el camino feliz, es decir, cuando todos los datos son válidos y esperamos que el Insert se realice correctamente, para eso vamos a usar para el identificador del usuario y demás claves foráneas , datos que se dieron de alta en la carga inicial, de esta manera estamos seguros que son válidos, para el resto de los campos vamos a definir algún dato que tenga sentido.

En el Expectedresponse vamos a poner el valor que devuelve el procedimiento en caso de éxito y para el mensaje de error, en caso de que falle el Assert, vamos a poner por ahora algo que nos permita identificar al caso que estamos corriendo, por ejemplo “caso 1”.

Esto del mensaje de error, es cuando hacemos el Assert, vamos a decir “bueno, queremos comparar un valor esperado contra un valor obtenido” y después le podemos poner un mensaje que queremos que aparezca en el caso que el Assert falle, ese mensaje puede ser cualquier cosa y lo podemos usar de la manera en que más nos convenga, en mi caso normalmente lo que trato de hacer, es identificar el “éste es el caso de prueba tal”, pero en realidad ese mensaje se puede usar como a uno le quede cómodo.

Ahora que cargamos datos para la prueba, podemos ejecutar nuestro Unit Test y para ello le vamos a dar

botón derecho y dar la opción Run Unit Test, cuando hacemos esto, GeneXus va a generar y especificar los programas necesarios para poder correr la prueba, al completar se abre una nueva ventana, el “Test Results Viewer” donde podemos ver los resultados de los Test que ejecutamos, en este caso nuestro Assert fue exitoso, es decir, el valor esperado es el mismo al valor obtenido y por tanto se muestra en verde.

Vamos a ver ahora un caso en el que pongamos datos inválidos y esperemos que la prueba falle.
(Más que que la prueba falle, lo que esperamos es que el resultado del procedimiento sea un error, para la prueba siempre esperamos que “pase”, que capture lo que esperamos)
Para ello, definimos una prueba en donde todos los parámetros de entrada al procedimiento que da de alta la queja van en blanco y el resultado esperado sería el mensaje que damos en caso de que hubiera un error, al mensaje que ponemos le vamos a decir que este es el caso en que si no hay datos, queremos que de un error.
Una vez que definimos este caso, lo que vamos a hacer es lo mismo que hicimos en el caso anterior, vamos a dar click derecho y ejecutar la prueba.

Para nuestra sorpresa, el resultado de la prueba no es exitoso y alguno de los Assert no se cumplió, si bien esperábamos que ambos funcionaran.
Para ver detalles de los Assert involucrados en una prueba, vamos a hacer click sobre la misma y abajo vamos a ver cada uno de los Asserts y sus resultados, lo interesante acá es que el Assert que no se cumplió fue el de la primera prueba, vemos aquí el mensaje Case 1, que fue el mensaje que asignamos a nuestro caso feliz.

Ahora bien ¿por qué la primera vez fue exitoso y ahora no?
Bueno, resulta que hay un índice único definido por descripción + usuario + fecha, por lo cual lo que está mal en este caso es la forma en que estamos intentando hacer la prueba, ya que dada la regla de nuestro negocio, no es posible dar de alta registros que tengan la misma información, por tanto vamos a hacer un truco para que nuestro Test funcione, que es agregarle algo que varíe en cada prueba, por ejemplo, vamos a ponerle un Time Stamp a la descripción.
Luego de esto, podemos volver a ejecutar las pruebas y ver que ahora sí, como esperábamos, nuestro Test es exitoso.

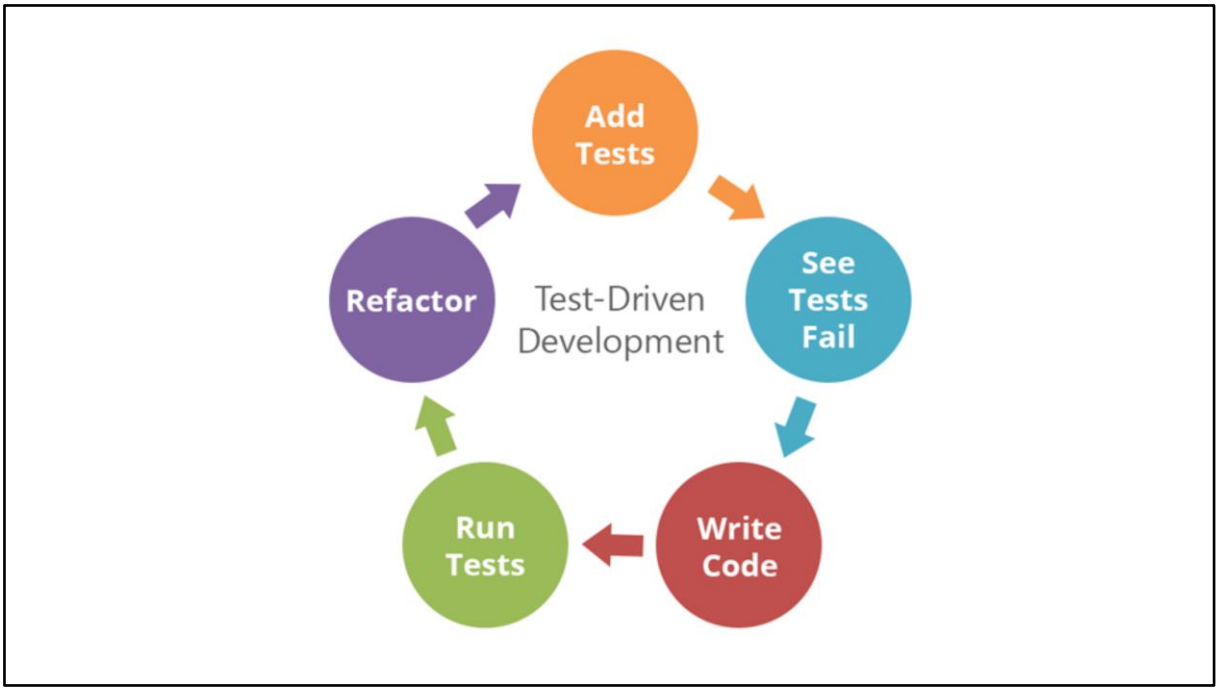
Vemos entonces que sencillo es realizar un Test Unitario y agregar casos de prueba.

Vamos a ver ahora un ejemplo en lo que sería el uso de pruebas unitarias usando técnicas de TDD, es decir, “Test Driven Development”.
Vamos a definir un caso en donde la descripción de la queja va vacía, el resto de los datos son válidos y decimos que esperamos que haya un error, es decir que si la queja o el reclamo no tienen contenido, esperamos que falle, al mensaje de error el mensaje en el caso del Assert, decimos que si no tiene descripción, debería dar error.

¿Vamos a ejecutar esta prueba a ver que pasa?
Nuestro Test no es exitoso y vemos que el caso que acabamos de agregar está fallando el Assert, si hacemos click en el botón de “+”, podemos ver detalles de cual es la diferencia y vemos que, esperábamos obtener un mensaje de error y sin embargo obtuvimos el mensaje de que el registro fue insertado exitosamente.

Decíamos que estamos haciendo prácticas de TDD, porque en realidad nunca se implementó en la transacción de reclamos las reglas necesarias para hacer estas validaciones, es decir, estamos haciendo el Test primero, antes de hacer la implementación.
Arrancamos entonces con un Test que sabemos que va a fallar y validamos que realmente falle y luego vamos a hacer la implementación, que en este caso consiste en agregar en la transacción Complaint, una regla que indique que el campo descripción no puede quedar vacío. Luego volvemos a ejecutar la prueba y vemos que ahora efectivamente la prueba es exitosa, de esta manera podríamos continuar el ciclo de TDD agregando escenarios adicionales, por ejemplo para validar que el domicilio tampoco esté vacío o que no

hubiera información respecto a la razón o el tipo de señalización.



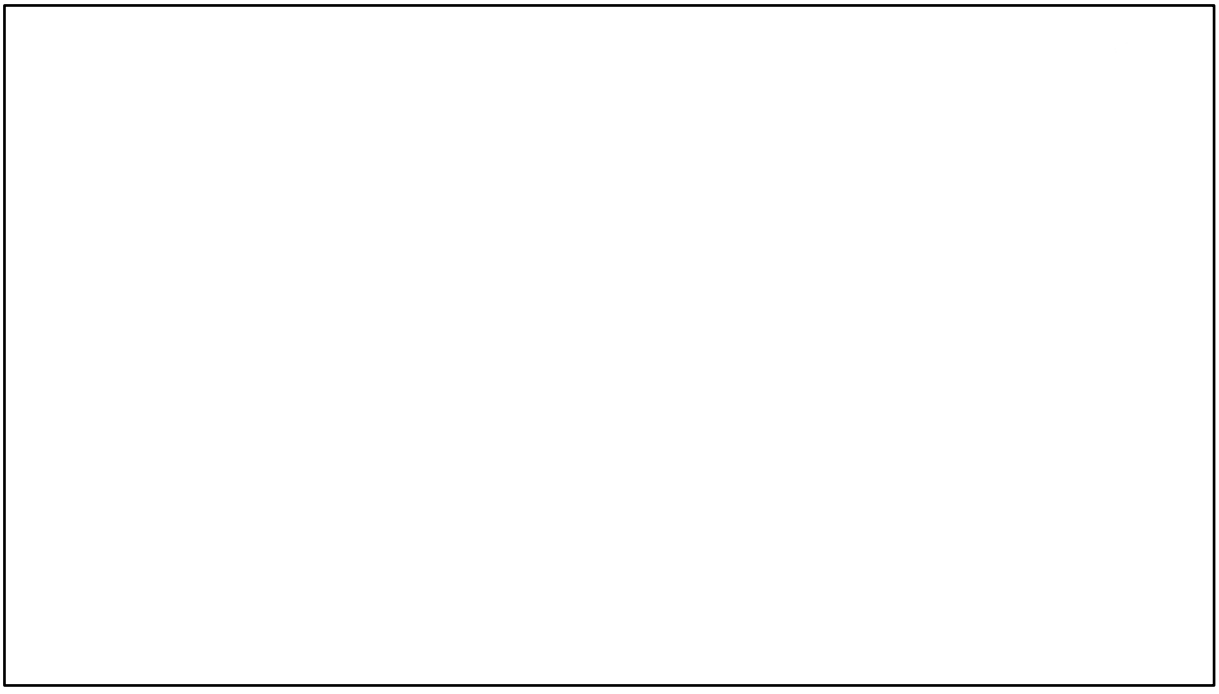
Este diagrama representa un poco el ciclo de TDD, donde agregamos primero un Test, el cual captura cuál es el comportamiento esperado del sistema, luego vemos que el Test Falla porque aún no está implementado, luego se escribe el código que implementa la funcionalidad esperada y se ejecuta el test para ver que ahora sí el mismo es exitoso.

Después se puede hacer un “Refactor” para mejorar la implementación.

Las prácticas de TDD dicen que la implementación que se debe hacer es la implementación mínima para que la prueba sea exitosa y luego sí podemos hacer el Refactor con más tranquilidad, sabiendo que tenemos una prueba que va a darse cuenta en caso de que rompamos la funcionalidad que habíamos implementado.

En definitiva, por las dudas, podemos trabajar de las dos maneras, podemos hacer TDD, o sea, hacer las pruebas primero, pero también podemos hacer las pruebas después en el momento que ya tengamos la funcionalidad implementada, para cualquiera de los dos escenarios esta funcionalidad nos sirve.

Otra cosa que ya que estoy aclaro ahora, es que estos ejemplos que estamos viendo son súper básicos, de repente no tiene mucho sentido hacer un Test para ver si un Bussines Component hace un Insert, para eso GeneXus ya hace sus miles de Test y su algoritmo que decide si salen a producción o no con la versión, pero lo que sí podría tener sentido es hacer los test que están validando si las reglas de la transacción se ejecutan de la forma que yo espero, porque ahí sí estamos evaluando nuestra programación de nuestras reglas de negocio.
Si lo hacemos de esta manera, nosotros podemos evaluar cómo se disparan las reglas y no necesitamos después ir a la pantalla y probar cada una de estas cosas a mano, entonces en realidad nos está ahorrando parte de lo que normalmente sería el Test manual.



Una cosa que es interesante, es que lo único que nuestra prueba está validando es el mensaje de respuesta del procedimiento que inserta, pero no sabemos a ciencia cierta si los datos quedan grabados o no en la base de datos.

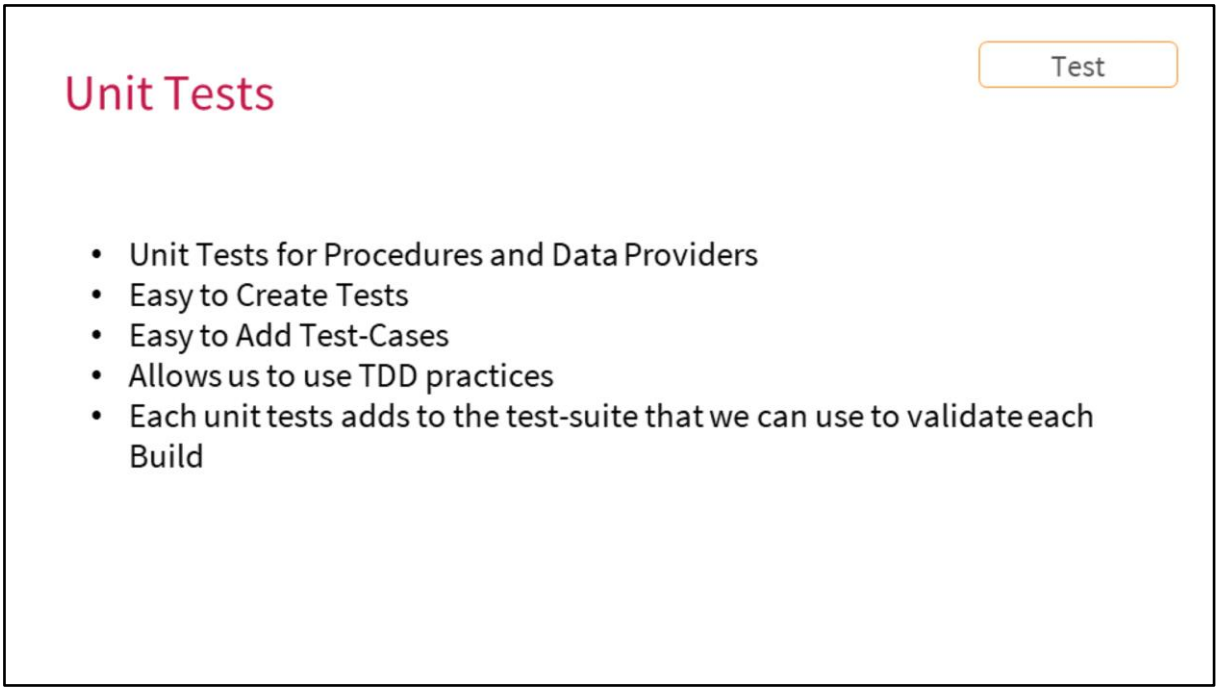
Vamos a ver cómo podemos agregar en nuestra prueba una validación de que realmente los datos estén en la base de datos.
Para ello, podemos agregar mas Assert en nuestro test, por ejemplo, yo aquí voy a agregar que se haga un Assert de los valores grabados en la base de datos para el reclamo que acabo de insertar.
En el caso de que la respuesta contenga el texto que se devuelve cuando hay un Insert exitoso, es decir "Thanks for…etc" voy a buscar en la base de datos haciendo un for each por la descripción, la fecha y el usuario, ya que sé que hay una clave única para ello. Y voy a hacer un Assert del resto de los atributos, viendo que contengan los valores esperados.
En caso de que no exista ningún registro que cumpla con la condición, voy a forzar un resultado de error a la prueba, haciendo un Assert forzado.

Eso del Assert forzado es un invento mío, no se si es parte de lo que en la herramienta pensaban, pero de alguna manera, si uno quiere poder disparar un error de algo que no está esperado, es una posibilidad.

Algo que es importante mencionar mientras ejecutamos las pruebas, es que existen 3 tipos de Assert:
AssertNumeric
AssertString
AssertBool
De manera que podamos más fácilmente comparar distintos tipos de datos.

Ahora que terminamos de ejecutar la prueba, vemos que el resultado es el mismo, es decir, sigue estando en verde (en exitoso) simplemente que tenemos más Assert porque ahora estamos validando también, que los datos estén grabados en la base de datos.

Hasta ahora hemos visto cómo ejecutar una prueba, pero también tenemos desde el menú Test la posibilidad de ver todos los test que tenemos definidos, en este caso tenemos uno solo y decidir desde aquí cuales ejecutar.
También tenemos la opción de ejecutar todas las pruebas con un solo click y eventualmente podemos ir también de forma rápida al visor del resultado de las pruebas.



Para repasar, hemos visto cómo podemos crear Test unitarios para procedimientos y Datas Providers, hemos visto que fácil que es crear estos Test, con botón derecho sobre el procedimiento que queremos testear y eso ya nos crea la estructura de la prueba.

Vimos que fácil que es agregar casos de prueba, modificando el Data Provider y agregando casos adicionales.
Vimos como es posible aplicar prácticas de TDD gracias a esta integración del Test Unitario dentro del ID de GeneXus.
Y otra cosa importante es que, los Unit Test forman parte de nuestra base de conocimiento.

Hasta ahora hemos estado ejecutando las pruebas a mano, pero es posible ejecutar los Test de forma automática como parte del proceso armado en nuestra integración continua.

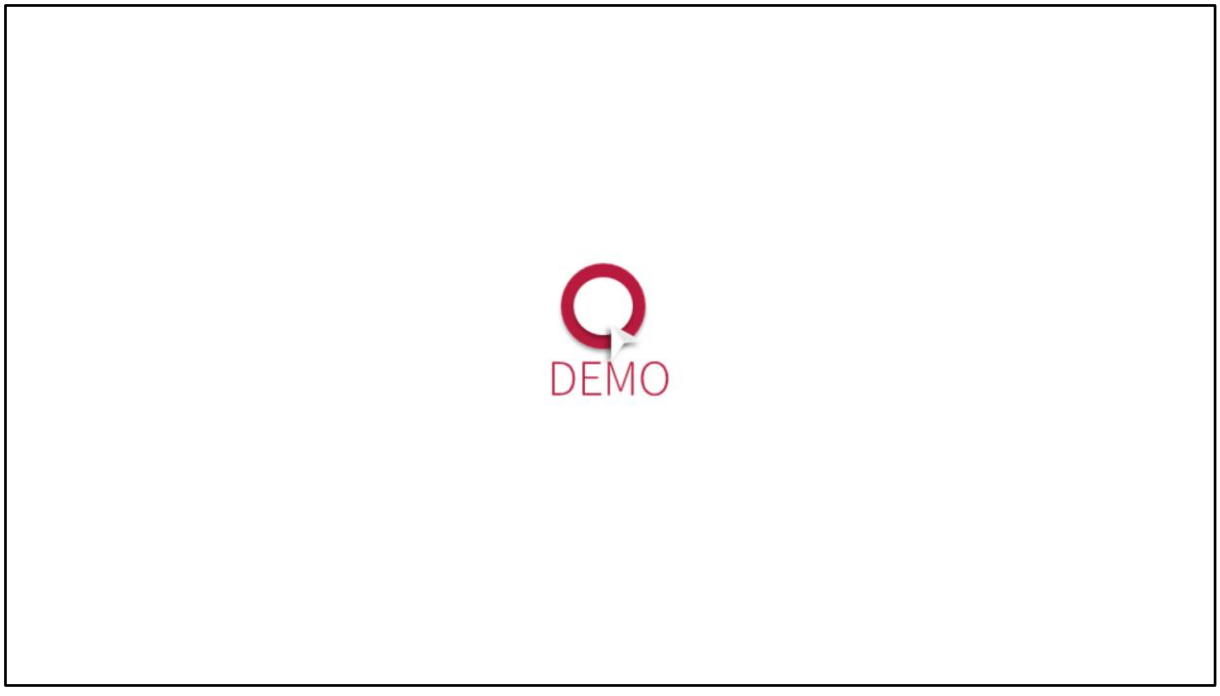
Ya que tenemos tiempo, aprovecho un minuto para decir unas cosas que iba pensando a medida que iba grabando esto o en realidad que iba haciendo estas pruebas, y es que en realidad cuando nosotros hacemos Test Unitarios nos vamos dando cuenta muchísimas veces que de repente nuestra implementación inicial de los objetos, podrían haber sido mejores o habernos dado más información, por ejemplo, si yo quisiera acá testear las diferentes reglas de la transacción, dado que el procedimiento Insert lo único que me devuelve es un mensaje que dice “error” “algo pasó” “intenta más tarde”, eso no me sirve mucho para validar si realmente está verificando que te falta la descripción, te falta el domicilio o lo que fuera, entonces me doy cuenta que quizás debería retornar algo un poco más interesante.
Las prácticas de hacer Test Unitarios, muchas veces nos ayudan también a ir desarrollando de forma que a la larga es más útil, que me va a servir para más cosas haber hecho esos procedimientos más encapsulados, con más robustez, etc.

- En el caso del retorno de los mensajes de error ¿no hay forma de capturar los mensajes que provee GeneXus? Como en el caso de los Bussines Component…
- Si, en realdad ahí el procedimiento en vez de devolver un mensaje escrito, podría haber devuelto el messages que el Bussines Component le devuelve.
- Pero ahí como que no es un Bussines Component, estas haciendo un procedimiento en el cual estas accediendo a una tabla y en esa tabla tiene asociado obviamente una transacción y los errores de esa transacción que salgan en el test… ¿me explico?
- No

- Cuando yo hago un Bussines Component y me da un error, me devuelve los mensajes de error. En el caso del Test, no trabajo en un Bussines Component, es un procedimiento pero me gustaría que me devolviera los errores que implican en la transacción.
- Por eso, a eso iba, en realidad ese procedimiento que es el que utiliza el Bussines Component, en vez del Response devolver un String, podría devolver el messages, podría devolver un código, podría devolver algo que a quien llama ese procedimiento, le diera más información para poder saber qué hacer y de hecho poderlo testear mejor.

- La consulta mía va en que yo veo que se generan muchos procedimientos para desarrollar el testing, cuando yo desarrollo y después quiero llegar a publicar mi aplicación ¿de alguna forma eso queda como un módulo aparte, queda empaquetado o lo deja al libre albedrío al desarrollador?
- Ahora mismo lo genera… (eso es algo que en futuras versiones seguramente se va a tener que mejorar) ahora cuando tu haces botón derecho y creas el test unitario, se crea en la misma carpeta en donde está el objeto que está testeando. En GeneXus son objetos distintos, de tipo Unit Test, no es un procedimiento y demás, es lo cual haría posible que después vos cuando hagas los Depoly Units o lo que sea para cuando vayas a hacer el Deploy los separes, pero en realidad ahora mismo quedan todos ahí. Y sí, efectivamente sería mejor que quedaran diferenciados.

-Una aclaración sobre el tema del Deployment, en realidad los Unit Tests que creas, llaman a tus procedimientos y a tus objetos reales y cuando tu haces Deploy en realidad tu marcas para hacer Deploy a algún objeto real tuyo, entonces los objetos creados para el Unit Testing no están en el árbol de Calls, porque en realidad son llamadores de tus objetos, entonces no están en el árbol y por lo tanto cuando haces Deploy no van en el Deploy, aun cuando estén en esas carpetas quizás, pero no son llamados, por lo cual no van.


DEMO

El proceso de Build forma parte del ciclo de desarrollo de la aplicación, por lo tanto la aplicación no termina solamente cuando se ejecuta y hacemos los Testing, sino que después de eso tiene que existir el despliegue de la aplicación en los servidores de producción.

Así que vamos a ver ahora algunas mejoras que incorpora GeneXus 16, mismo en el proceso de Build y también después en el proceso de puesta en producción

Package/Release/Deploy

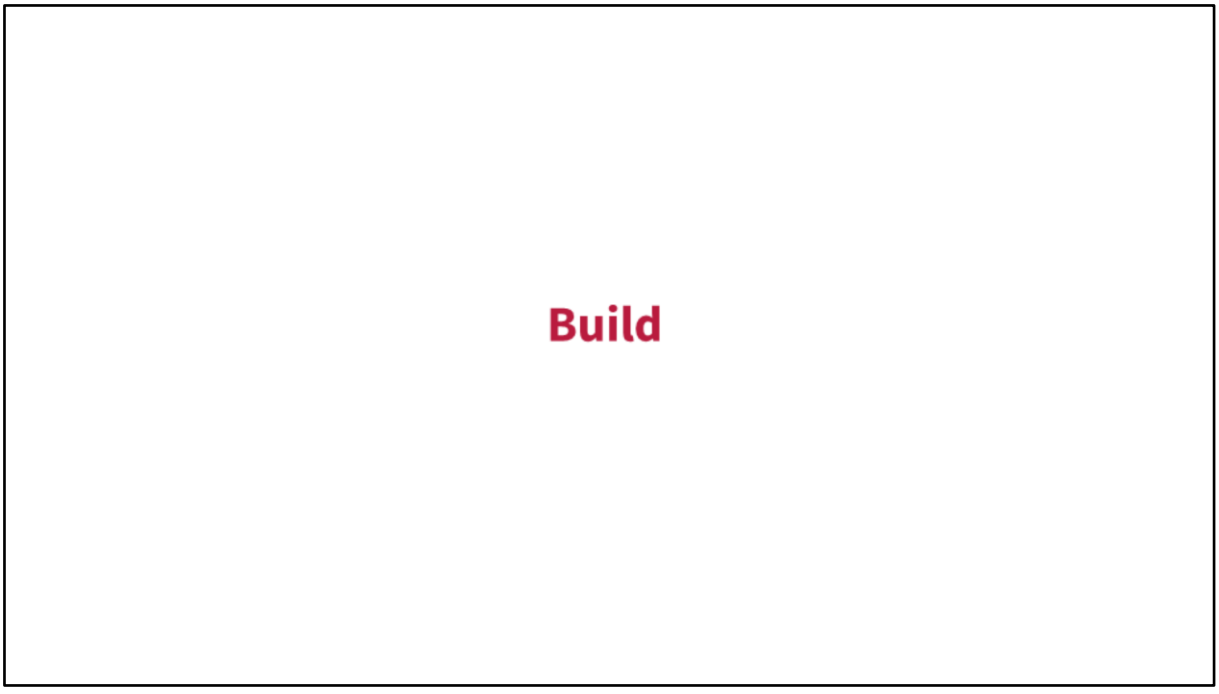
Deployment Unit Object
Build events
Serverless (deploy to AWS lambda)
GX Cloud Deployment Services → F6 PAAS
Containers (Docker)

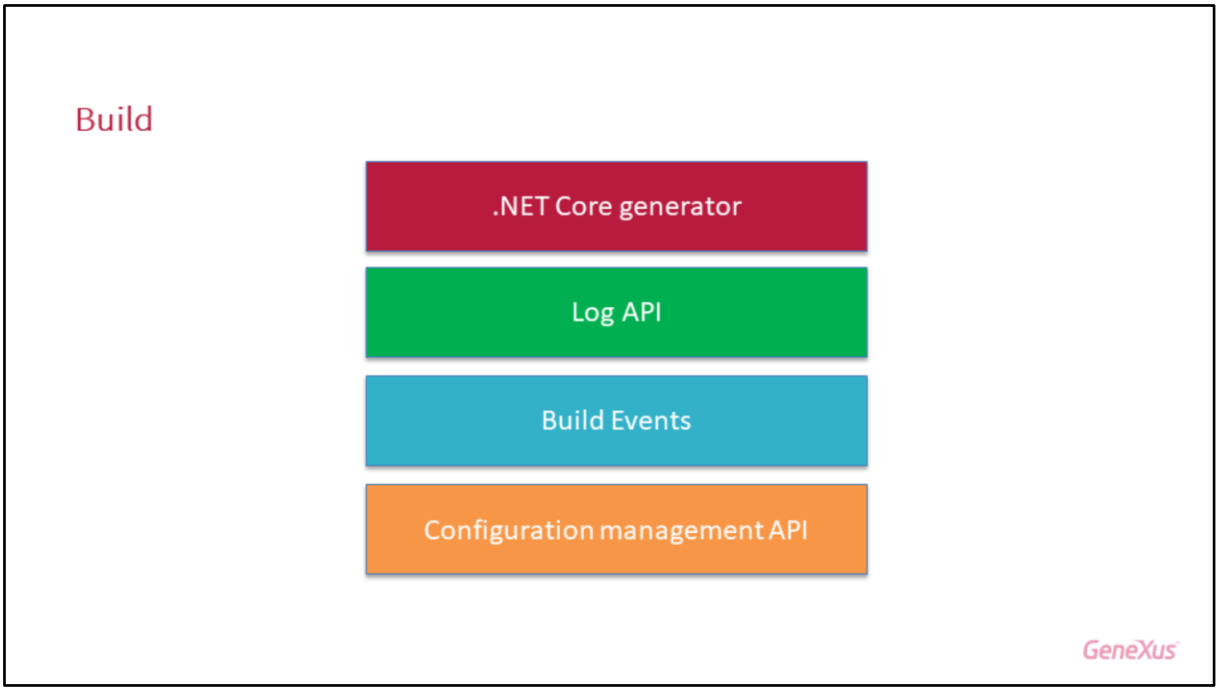
Configure/ Operate

ConfigurationManager API
Config via Environment variables

Monitor

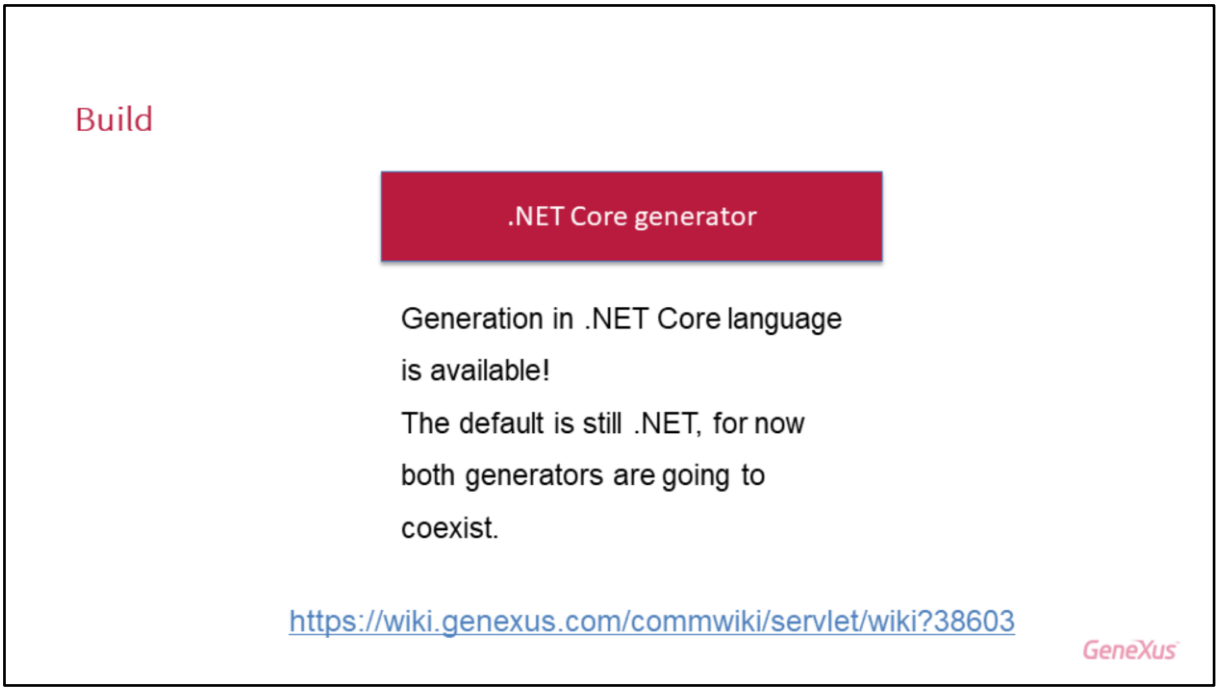
Log API


Build



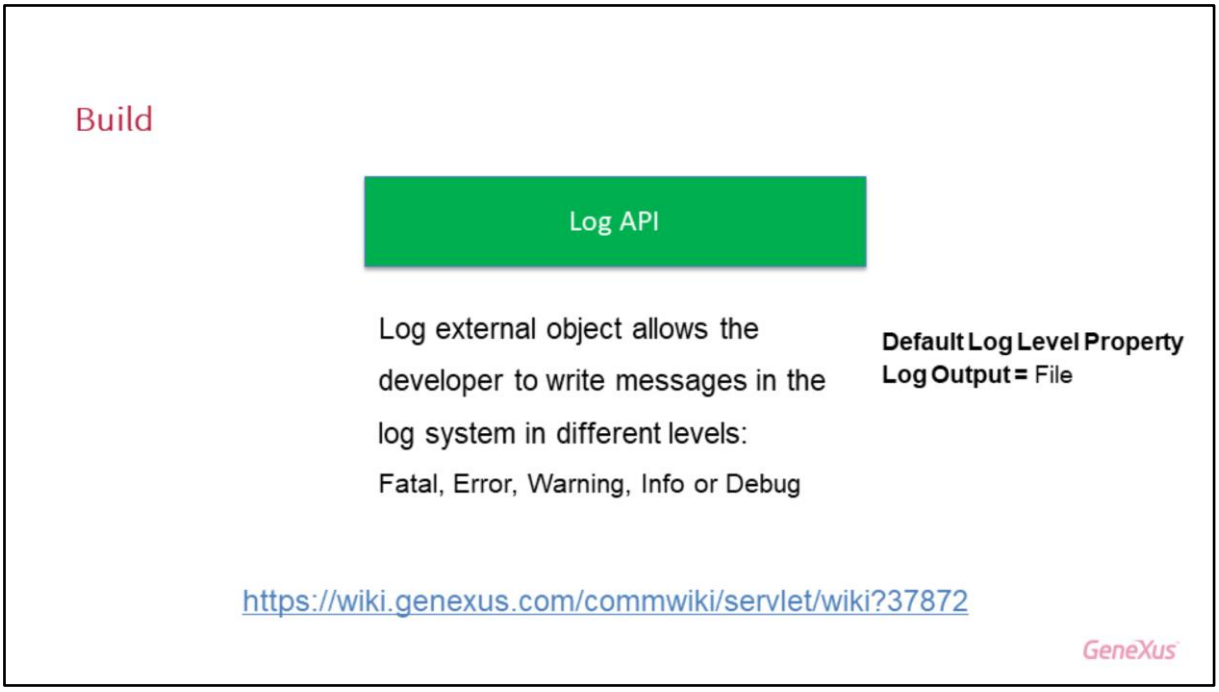
Veamos primero la parte de Build.

Vamos a ver que incorporamos un nuevo generador, el generador .Net Core, este generador no sustituye al generador .NET que tenemos ahora, los dos van a seguir coexistiendo, pero vamos a ver algunas características de esto.
Después vamos a ver también la posibilidad de que el usuario pueda crear sus propis Logs, los Logs del sistema mediante una API que se llama Log API.
Vamos a ver también eventos de Build, o sea que que vamos a poder ejecutar tareas inmediatamente antes de un Build o inmediatamente después de un Build con la posibilidad de la codificación de estos eventos.
Y en la parte de Build por último, también vamos a ver una API que nos permite leer la configuración de la aplicación en ejecución, aquella configuración que se guarda en variables de ambientes o en los archivos, lo vamos a poder ver a través de una API.



Bueno, el generador .NET Core es la nueva versión de generadores que tiene Microsoft, que probablemente algún día sustituya al generador .NET, pero no todavía.
Es un generador multiplataforma de código abierto, quiero decir que es compatible con Windows, con Linux y con Mac.
Es un generador que sirve para generar aplicaciones Web y también aplicaciones de consola o aplicaciones en el Server, quiere decir que es una herramienta bastante completa pero que por ahora Microsoft tiene la estrategia de que siga coexistiendo, por esa razón nosotros en GeneXus también, si bien podemos generar con .NET Core, vamos a seguir teniendo el generador .NET, los dos generan C#.

Así que ya tenemos la posibilidad de poder generar con este generador.
En el Wiki tienen el link de lo que se puede utilizar, pueden ampliar en esto.

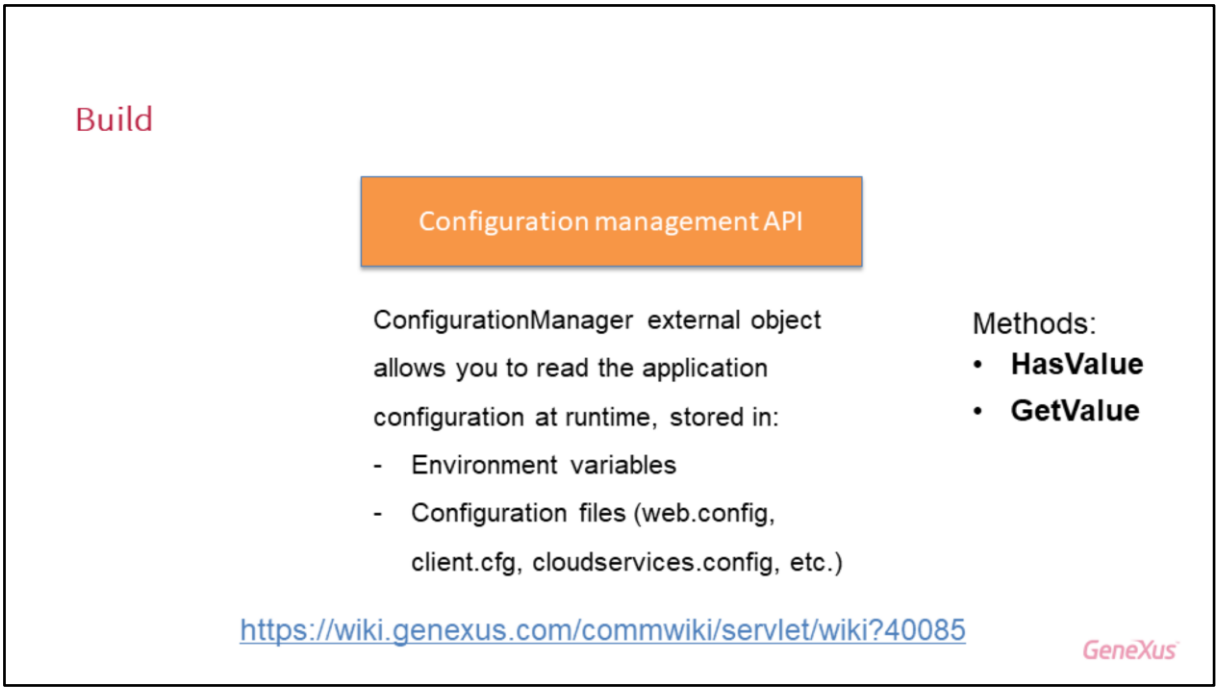


Log API, hasta ahora el usuario teniendo las propiedades, por ejemplo Default Log Level prendida y el Log Output = File, nosotros podíamos setear distintos niveles de Debug y podíamos hacer que se escriba en el Log, pero siempre era el Log del sistema, ahora con esta API nosotros vamos a poder incluir nuestros propios mensajes de error, que salgan incluidos dentro del Log en el sistema, en los distintos archivos del Log del sistema.
Inclusive podemos categorizar esos mensajes de error de distintos tipos, es decir que nosotros podemos crear nuestros mensajes y catalogarlos ya de entrada como mensajes “fatal” o “error”, “warning”, “info” o “Debug” y van a salir con esa categorización dentro del archivo del Log del sistema.

En el caso de .NET, es el “Client.log” y en el caso de Java se guarda en los distintos archivos Log que tiene el Tomcat, que divide el Log en varios archivos.
Así que vamos a poder nosotros desde nuestra aplicación generar entradas de Log y vamos a poder verlos en el Log del sistema.

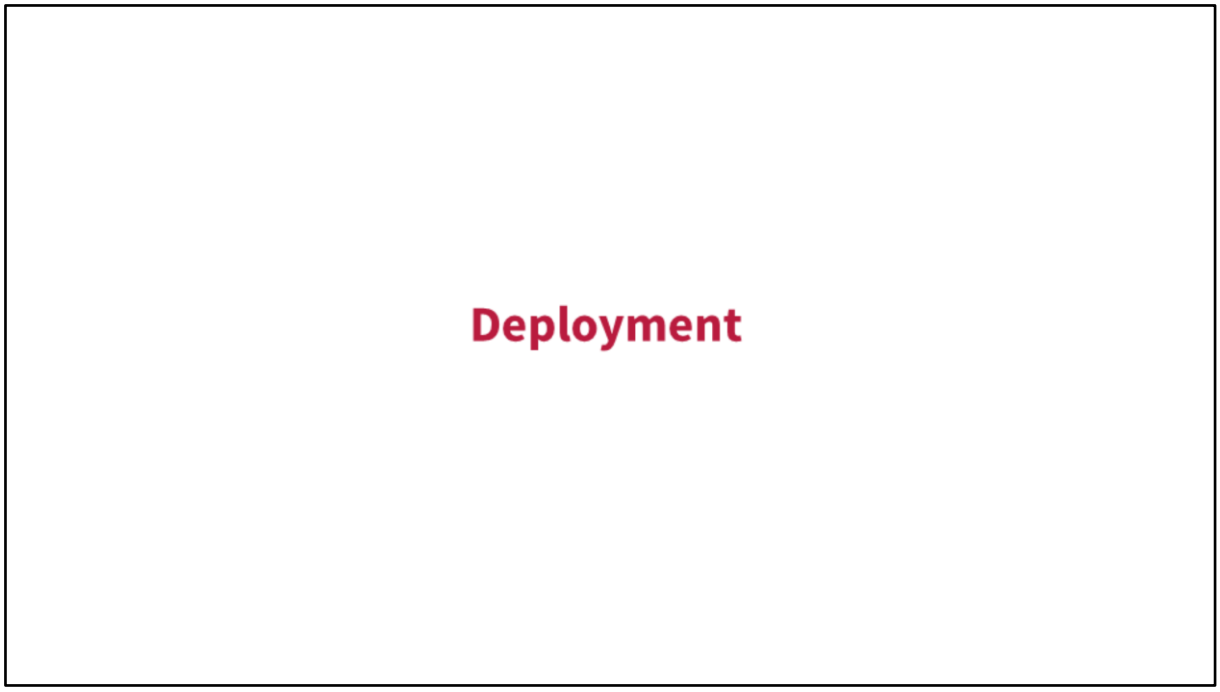
Esto es válido no solamente para Web, sino también para SD
En el caso de Android está la herramienta LogCat que nos permite ver los Logs e inclusive se puede invocar desde el Android monitor.
Y en el caso de que estamos generando iOS, podríamos verlo en la consola de Debug el compilador XCod

Tenemos la posibilidad con GeneXus 16 de hacer algunas pequeñas tareas antes del Build o después del Build, siempre pasaba que muchas veces teníamos que copiar algunos archivos o que teníamos que limpiar un directorio o hacer algunas tareas sencillas que queríamos que se hicieran antes de que se produjera el Build o inmediatamente después que se hiciera el Build.
Ahora eso lo podemos hacer mediante la utilización de eventos de Build, para eso utilizamos el menú de Build y Build Events, ahí vamos a tener un menú donde ya hay codificados algunos marcos que nos va a permitir hacer algunas operaciones básicas o como por ejemplo, copiar archivos o limpiar un directorio, etc. Así que ya lo tenemos disponible desde allí.

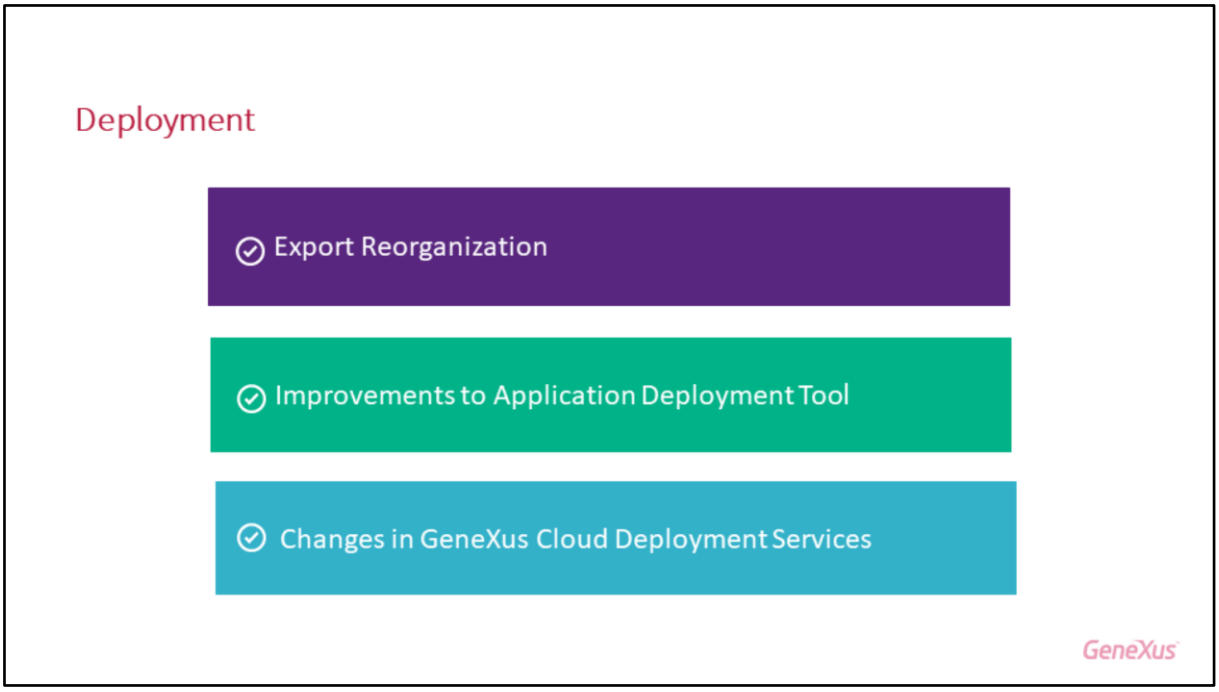


Bueno, otra cosa que se incorpora en la versión 16, es una API que nos permite ver qué es la configuración que se está utilizando en la ejecución de la aplicación.

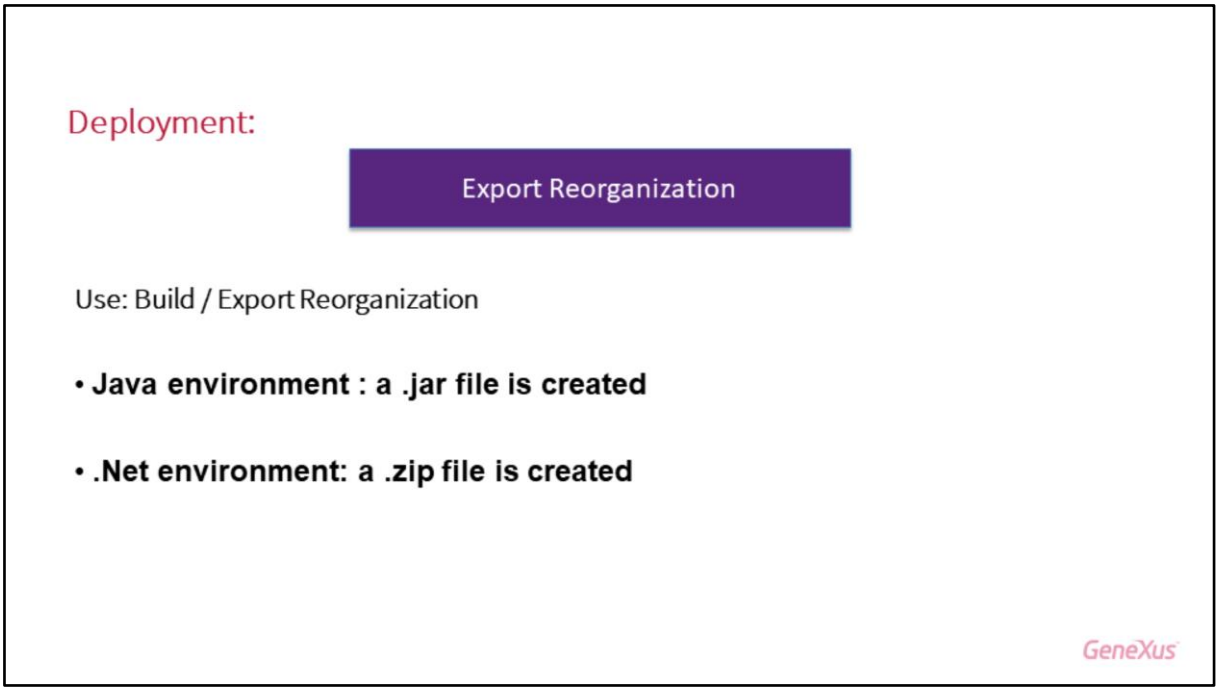
Esa configuración normalmente se guarda en los archivos “Web.config”, “Client.cfg” o “Cloudservice.config” y eso normalmente lo generaba GeneXus, pero muchas veces para determinado servidor o para determinada aplicación lo tocábamos a mano en esos archivos y ejecutábamos con determinada versión nuestra de esos archivos, pero después que estaba ejecutando no sabíamos exactamente cuales eran los valores que estábamos tomando y teníamos que ir a mirar el archivo de vuelta para saber como era que estábamos ejecutando, bueno, ahora desde Runtime, desde ejecución, vamos a poder acceder a los valores de esos archivos y poder obtener información de cual es efectivamente la configuración que se está utilizando para la ejecución.
No solamente en cuanto a los archivos estándares (que son esos) sino también a la variable de ambiente porque para varios entornos, por ejemplo en JAVA, se setean variables de ambiente y ahí se guardan determinados valores, así que también desde ejecución vamos a poder consultar esa variable de ambiente para saber cómo es que está corriendo nuestra aplicación.
Inclusive se pueden utilizar también otros tipos de archivos y se pueden incluir dentro de invocaciones a la API, la API básicamente tiene dos métodos, el método HasValue y el método GetValue que nos permiten ver si existe un determinado parámetro, si un determinado parámetro de configuración esta seteado (con el HasValue) y después con el GetValue, obtener el valor que tiene ese parámetro dentro de los archivos o en la variable de ambiente.



Veamos ahora algunos cambios que hay en la parte de Deployment.



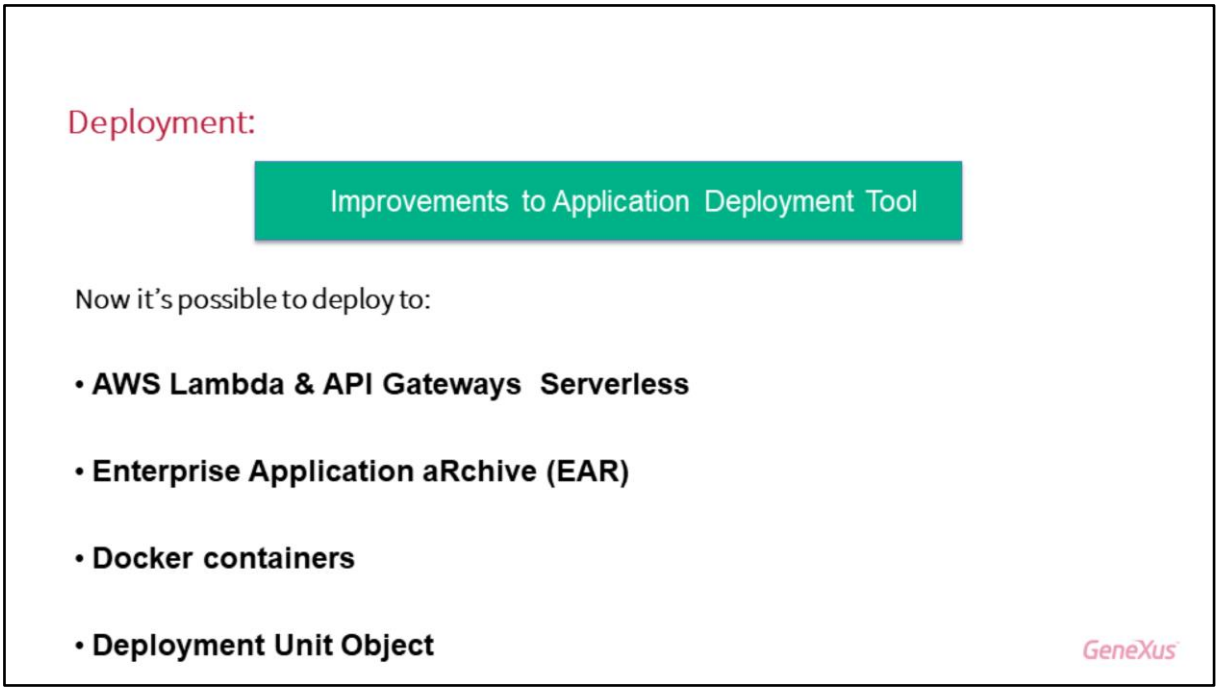
Básicamente tenemos tres cosas disponibles.
Ahora se va a poder hacer un Export de la Reorganización, también vamos a ver algunas mejoras que hay en la herramienta de “Application Deployment Tool” que es la aplicación que usamos ahora para poder despegar las cosas, que ya viene desde la Evolution 3 (últimas upgrade) y algunos pequeños cambios que hay también en la Cloud Deployment Services, sobre todo en la parte del F6



En cuanto a la exportación de la Reorganización…
Antes se generaba siempre un "reorg.exe" con los cambios de la ultima reorganización y nosotros lo que teníamos que hacer era ir y copiarnos ese reorg.exe a nuestro servidor en producción y después ejecutarlo, con lo cual se hacían los cambios desde la última reorganización que se había hecho.
Ahora haciendo Build, Export Reorganization, lo que tenemos es la posibilidad de empaquetar todo lo necesario para que esa reorganización se ejecute, entonces, por ejemplo, en el caso del Java se genera un "jar" con todo lo necesario para la reorganización , eso no cambió mucho respecto a lo que había en cuanto a Java, pero en .Net ahora lo que tenemos es un Zip donde se empaqueta la Reorganization.dll y el Reorg.exe y eso se puede llevar al Server y ya lo podemos utilizar directamente allí, pero siempre teniendo en cuenta de que es desde la última reorganización.
Si es la primera vez que se hace la reorganización, se van a crear las tablas y todas las estructuras en la base de datos, pero si es una reorganización posterior, se va a hacer solamente desde los cambios de la última reorganización, así que igual de todas maneras vamos a tener que tener el cuidado del versionado, tener en cuenta cuales fueron los cambios que se habían hecho.

- Hola, ¿está implementado tarea MSBuild para esto del Export Reorganization?
- Sí, está y también una cosa que le estaba comentando a Cecilia, está para .Net Core

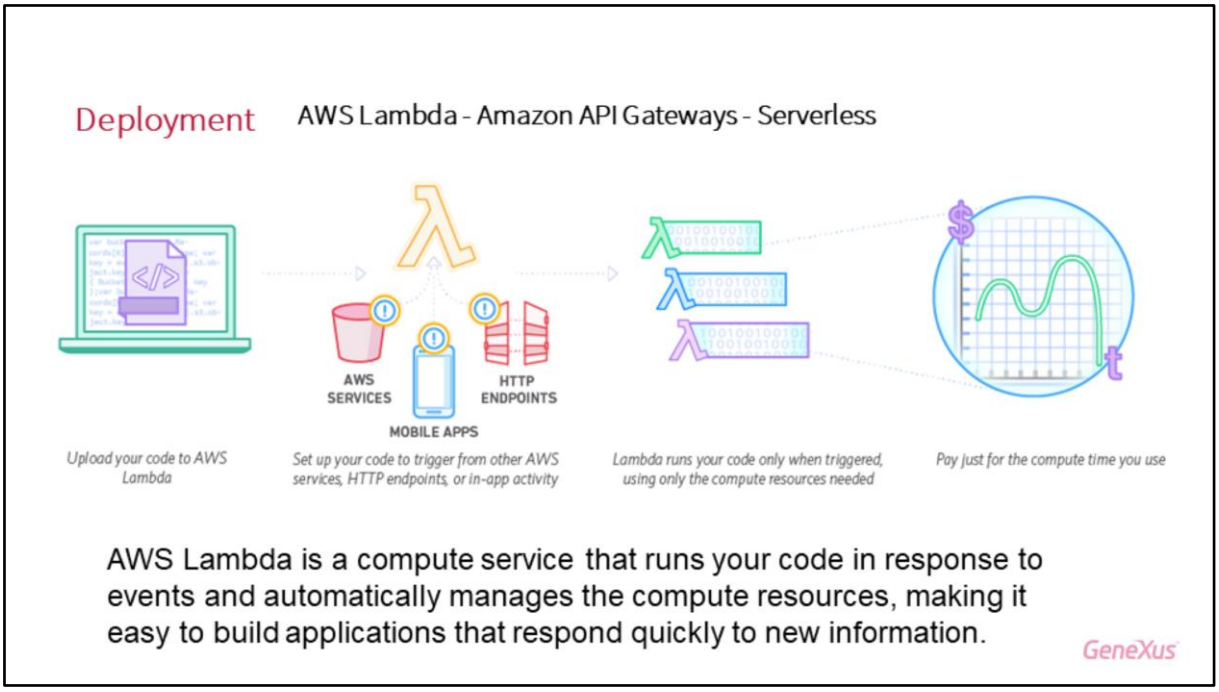
- Una consulta, con respecto a dentro del empaquetado en el caso de .NET, por ejemplo ¿van todas las carpetas de Reorganización? ¿Solamente el "Bin" o por el tema de que van los fuentes y todo eso?
- No, no! Acá solamente estamos hablando de lo necesario para hacer la reorg en la base de datos, o sea, lo que va es la Reorganization.dll y el reorg.exe
- Ok, gracias!



En cuanto a la Application Deployment Tool que es la herramienta que tenemos ahora para hacer Deploy de aplicaciones, recuerden que antes, hace años teníamos el Deployment Wizard que nos permitía generar un War cuando hacíamos un Deployment en Java, pero no teníamos una herramienta para hacer Deployment en .Net, hubo sí en algún momento una extensión que nos permitía guardar todo eso en una carpeta y después eso lo podíamos llevar al Server, pero no había una herramienta que podía empaquetar todo eso.
Bueno, a partir ya de la versión 15 (en realidad desde la Evolution 3, últimas versiones) teníamos disponible esta aplicación, pero en la V16 tenemos algunas ventajas y algunos cambios, como por ejemplo, vamos a poder utilizar los servicios de Lambda, de Amazon Web Services que nos van a permitir responder a eventos de nuestra aplicación, que el servicio va a poder hacer que se responda eventos de nuestra aplicación, que se realicen. Y lo mismo pasa con los API Gateways que van a ser los servicios rest, de front-end de esos servicios y todo esto sin preocuparnos nosotros de tener que poner un servidor, instalar todo el Software… sino que esto es "serverless", están los servicios disponibles, usamos solamente los servicios y no nos preocupamos de hacer toda la instalación del server en la nube.

Después también tenemos la generación de archivos EAR, para cuando estamos generando un Java Bin. Vamos a ver también que ahora vamos a poder desplegar en conteiner de Dockers, que es bastante bueno en el sentido de que uno va a poder generar un despliegue en algo que es parecido a una máquina virtual pero mucho más liviano, porque solamente se van a estar virtualizando la parte de Software y toda la parte de Hardware no, eso lo vamos a aplicar después, las dos últimas partes están en un video que incluye después una Demo con utilización de Dockers y haciendo el despliegue en Docker y viendo como ejecuta.

Por último, la Deployment Unit Object, nos va a permitir agrupar despliegues en un objeto nuevo que se llama Deployment Unit.
Muchas veces pasaba que cuando se quería hacer un Deployment se quería hacer Deployment de determinada parte de la aplicación, por ejemplo, solamente de los servicios o solamente de la parte Web, pero no teníamos una forma de poder guardar eso para reutilizarlo después.
Ahora tenemos disponible la Deployment Unit que es un objeto que nos permite arrastrar los objetos main y todos los objetos que queramos incluir en ese despliegue en particular y después vamos a poder hacer despliegue solamente de eso, así que vamos a poder agrupar en distintas Deployment Unit las distintas partes de la aplicación para que queden mas ordenados los despliegues y poder utilizarlos después.



Lo que serían los servicios de AWS Lambda – Amazon Web Services Lambda:

Nuestra aplicación cuando está en el servidor, siempre tiene que responder a ciertos requerimientos de clientes, por ejemplo, el cliente quiere actualizar un valor, quiere solicitar un dato o quiere un sensor que esté enviando información al servidor y quiere que ese dato se guarde, hay un montón de eventos que se producen en nuestra aplicación a nivel del Server que la aplicación tiene que responder a esos eventos.

Hasta ahora nosotros teníamos que preocuparnos nosotros de responder a esos eventos y hacer además toda la administración del Server, es decir que si la cantidad de accesos concurrentes de muchos clientes a la aplicación se subía, íbamos a tener que nosotros proveer el adecuado escalamiento del servidor y tendríamos que además preocuparnos de todo lo que le poníamos en ese servidor, si había patches, o si había upgrades o todo lo demás… teníamos que preocuparnos nosotros.

Ahora con los servicios Lambda, tenemos la posibilidad de escribir un cierto código y que se encargue el Lambda de hacer que cada vez que haya un evento se ejecute el código que nosotros pongamos para responder a esos eventos, sin que nosotros tengamos que preocuparnos de monitorear el servicio y tampoco preocuparnos de hacer los escalamientos, o sea que, el mismo servicio Lambda ejecuta el código que nosotros pongamos, ese código puede ser ya sea de Aplicaciones Mobile, ya sea de aplicaciones IOT o ya sea de aplicaciones Web, o inclusive algunos eventos que son propios del servidor, como por ejemplo algo que se haya schedulado y que tenga que ejecutarse, todos esos eventos son respondidos por el Amazon Lambda en función a los requerimientos que se tengan y ejecuta el código nuestro.

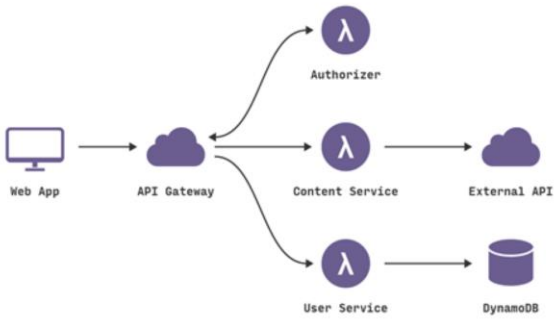
Otra cosa que tiene importante, es que en realidad solamente nos van a cobrar…en lugar de cobrarnos por el servidor cuando nosotros instalamos un Server, por ejemplo un “Platform as a Service” y le ponemos todo el Software, entonces nos cobran un cierto dinero fijo por tener eso disponible, en este caso el servidor lo provee Amazon Web Services a través del servicio (lo que estamos haciendo es consumiendo servicios) y solamente nos van a cobrar por cada vez que se usen esos servicios, es decir, cada vez que hay un evento y ese evento se responde con un código, (que lo responde el servicio Lambda) nos cobran solamente por el tiempito en que se utilizó eso, entonces esa función ServerLess en realidad es mucho más económica y además no nos tenemos que preocupar de todo lo que es el mantenimiento del Software, de los Upgrades y de todo lo demás que sería necesario si la administración del Software estuviera a nuestro cuidado.

Participantes:

- ¿Tienen pensado hacer algo para el mundo Assure de forma similar como las “functions” o “Web Shopp” que también corren de esa misma manera?
- Sí, tengo entendido que sí, que todavía no está disponible, pero sí, está dentro de las líneas… de hecho, en todas las plataformas en que nosotros tenemos servidores en la nube, la idea es tratar de proveer servicios similares a estos, siempre y cuando el proveedor los tenga.

Deployment

AWS Lambda - Amazon API Gateways - Serverless



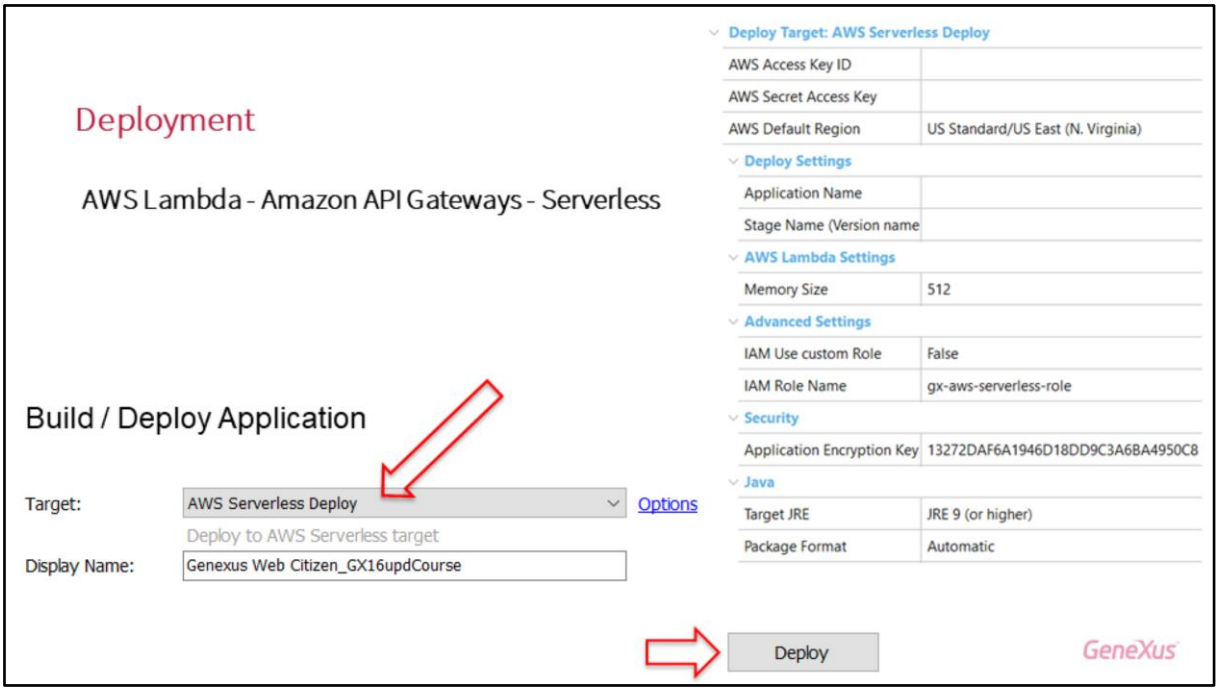
Benefits:

- Integrates with AWS Lambda to allow you to create completely serverless APIs
- You create REST APIs that let you call publicly available AWS services through your code running in AWS Lambda
- You pay only for calls made to your APIs and data transfer out.

GeneXus

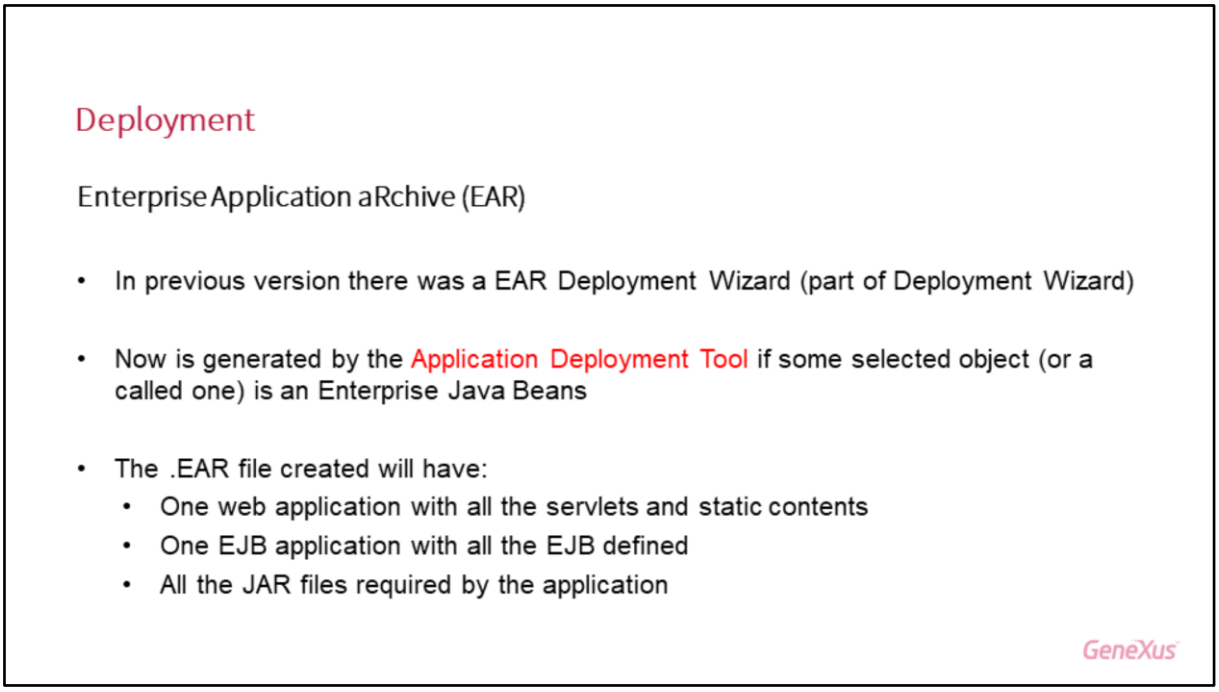
Cuando se usan los servicios Lambda y lo que quiere acceder al servicio es una aplicación Web, se necesita que haya cierta interface Rest con la cual la aplicación Web pueda interaccionar con esos servicios Lambda, entonces para eso existe la API Gateways, que nos permite que la aplicación Web tenga una interface de entrada, ciertos servicios Rest publicados que en los cuales nosotros podemos invocar para poder acceder a los servicios.

También de la misma forma, solamente se paga por cada vez que la aplicación Web accede a través de la API GateWay a los servicios de Lambda y de la misma manera todo esto estaría dentro de lo que sería ServerLess.
Lambda además puede acceder a otros servicios propios que tiene Amazon, como el Dynamo DB u otras external API y la “puerta de entrada” a todo eso, es una interface Rest que está dada por los API GateWay.



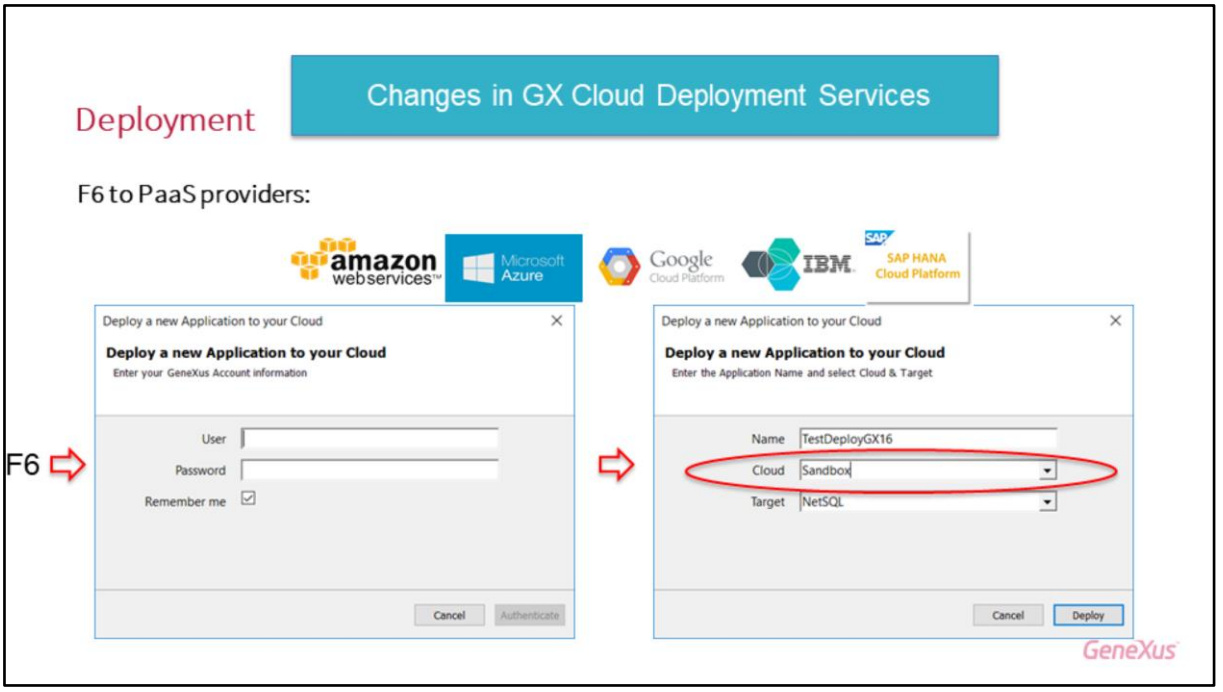
¿Cómo usamos esto desde GeneXus?

Lo que hacemos es, mediante Build, Deploy application y lo que elegimos acá es el Amazon Web Service ServLess Deploy y lo que hacemos es configurar las Keys, las credenciales que nosotros tengamos para poder acceder a ese servicio Lambda que nosotros contratemos, y después simplemente al hacer Deploy vamos haciendo el Deploy utilizando el AWS Lambda y en el caso de aplicaciones Web, la API GateWays



Si lo que nosotros tenemos en nuestra aplicación Java es un archivo de tipo Javabeans, lo que teníamos antes era un Wizard para generar el Enterprise application archive EAR, ¿se acuerdan?, que se abría un Wizard especial para eso cuando estábamos en el Deployment de Java, bueno, ahora eso en realidad está integrado dentro de la Application Deployment Tool, así que basta con que nosotros arrastremos un Javabeans a la ventana de Application Deployment Tool para que se genere el archivo EAR correspondiente.

Ese archivo EAR que se crea va a tener la Application Web, los ServLess, todo el contenido estático, todos los “jar”, es decir, todo lo que se necesita para que quede funcionando ese archivo Javabeans que habíamos arrastrado a la Deployment Tool (esto no estaba disponible y ahora ya está integrado dentro de la Deployment Tool en GeneXus 16)

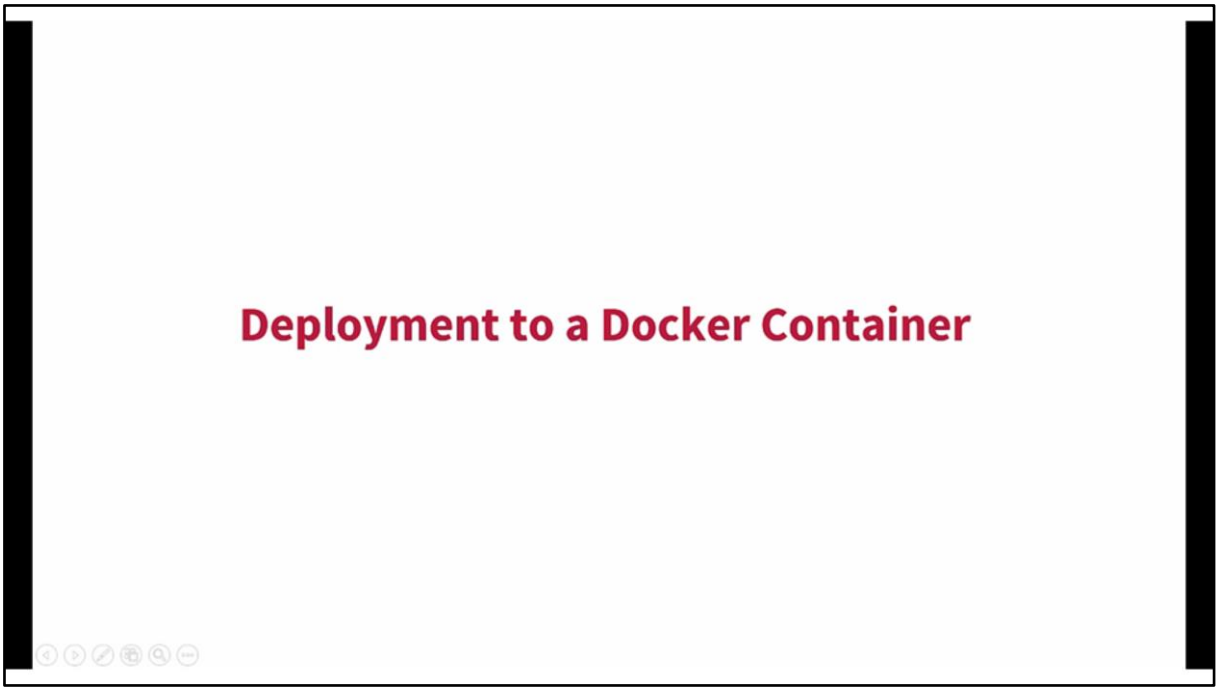


Un cambio más , cuando nosotros hacemos F6 lo que estamos haciendo es un Deployment al servicio que tenemos contratado en algunos de los proveedores de the Cloud, que para eso, previamente GeneXus nos pudo ayudar en armarnos el ambiente necesario para que nosotros podamos hacer Deployment a la nube, o también si lo quieren lo pueden armar ustedes, en el Wiki está toda la documentación como para hacerlo, pero en realidad es más fácil y no es para nada caro solicitar los servicios de GeneXus para que los ayuden a preparar todo el ambiente y ya quede pronto para hacer el Deployment.

Entonces, cuando hacemos F6 lo primero que se va a pedir es las credenciales de la GeneXus accounts, para poder tener ustedes permiso y poder hacer ese Deployment a la nube, a cualquiera de las plataformas y después se va a elegir en este cuadro de diálogo cuál es el Target, cuál es el environment que ustedes están desplegando y cuál es en el Cloud aquel servicio que está preparándose, ahí les van a salir los nombres de los servicios o paquetes, configuraciones que ya están armadas (que armó GeneXus para ustedes) o si no tienen ninguna, van a tener posibilidad de hacer un Deployment en el Sandbox para probar, eso cada vez que hacen F6.

F5 nos permitiría hacer varias ejecuciones como hasta ahora, pero cuando hacemos F6 sería una puesta en producción y una subida automática a los servidores de alguna de las plataformas de Cloud.

La diferencia que tiene la versión 16, es que hasta ahora esos servicios que estaban en la nube eran del tipo IaaS (Infrastructure as a Service) pero ahora también pueden ser Plataform as a Services, eso viene a partir de la versión 16.



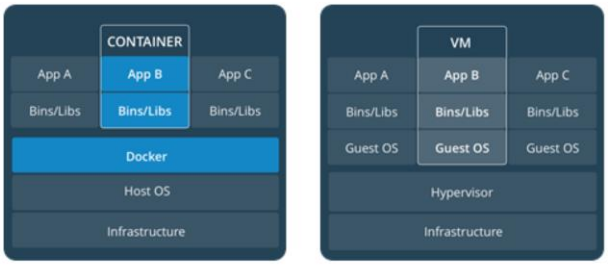
Veamos ahora un concepto nuevo, es el Deployment a un Docker Container.

Esto tiene que ver en similitud al caso de los barcos, los barcos cuando llevan carga… la carga antes se llevaba suelta y se iba acomodando en la bodega de acuerdo al tipo de carga, si una carga se estropeaba o se rompía, que no estropeara la carga que estaba al lado, entonces se agrupaba por tipo de dimensiones y por tipo de carga, etc.
Luego se crearon los contenedores de ciertas dimensiones y ciertas características, y hoy en día todo el transporte se hace con contenedores.
Hoy en día en base a esas medidas estándares (que son internacionales) hasta los mismos barcos que transportan los contenedores, las grúas que las manejan en los puertos… todo está estandarizado para manejar ese tipo de contenedores, entonces, aquí viene la idea de poder utilizar contenedores para poder desplegar Software.
La idea de usar contenedores para empaquetar aplicaciones de Software empezó en Linux que creó los Containers, pero eran difíciles de usar y requerían de conocimientos avanzados de las funciones de kernels de Linux, es decir, el desarrollador tiene que ser un experto en Linux y tiene que saber qué funciones tocar del Kernels directamente como para poder utilizar esos Containers.

La empresa Dockers, creó una serie de servicios que acceden al Kernel y lo disponibiliza como una API, esto logró un nivel de abstracción mucho mayor y facilitó su uso, permitiendo que se hiciera popular el uso de los Conteiners para hacer despliegue de aplicaciones.

Docker

- Platform to develop, deploy, and run applications using containers



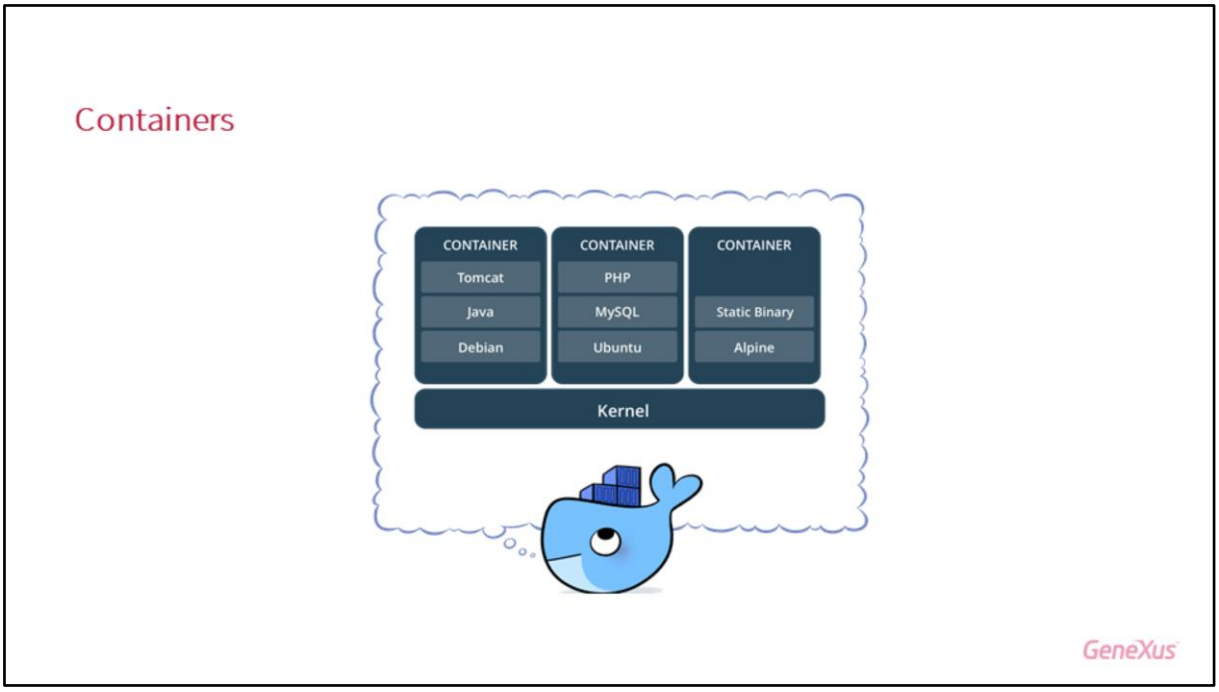
GeneXus

Veremos ahora los conceptos principales que hay dentro de esto:

Docker es un producto, una plataforma, pero también es el nombre de la compañía (Docker Inc.)

Docker Inc. creó una plataforma llamada Docker que facilita el desarrollo, despliegue y ejecución de aplicaciones en contenedores.
Esta plataforma provee una virtualización que hace que la aplicación pueda ejecutar dentro del Container, con todos los recursos que necesita, y además puede ejecutar en cualquier computador.

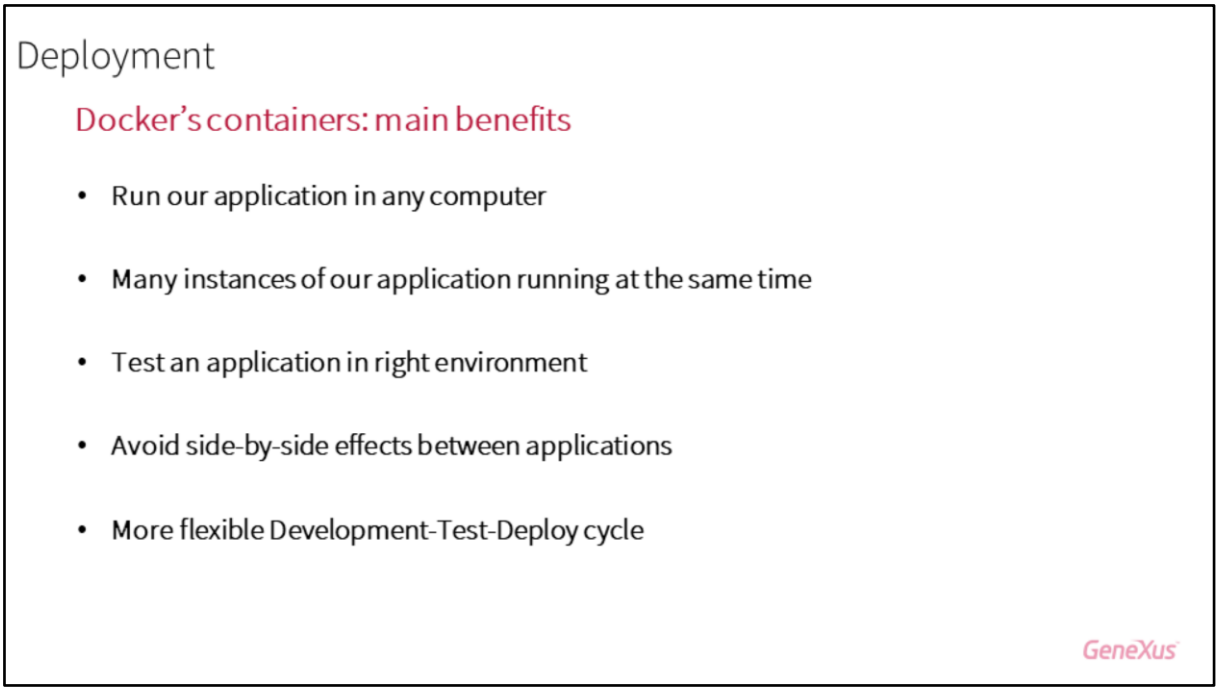
Uno podría pensar que un contenedor se parece a una máquina virtual, pero no es así, una máquina virtual replica también el Hardware (toda la parte física), mientras que el contenedor ejecuta sobre el Hardware existente y usa solamente las funciones del Kernel del sistema operativo Host.
Si vemos en el diagrama, aquí vemos el caso de una máquina virtual donde tenemos aquí el sistema operativo, donde está la parte de Hardware (de infraestructura), la parte del sistema operativo, todo eso es emulado por la máquina virtual y después encima de eso estarían los binarios, las bibliotecas, todo lo necesario y nuestra aplicación.
En cambio, Docker genera una capa que se comunica directamente con el Host del sistema operativo, esa capa es generalmente en sistema operativo Linux, pero esa capa provee APIs y nos permite ejecutar un contenedor no solamente en Linux, sino también, por ejemplo en Windows.
Ven que aquí permiten que el contenedor incluya solamente la aplicación, las librerías y todo lo que se necesita, pero no requiere de que se vuelva a virtualizar el sistema operativo o el Hardware como en el caso de la máquina virtual, por lo tanto es mucho más eficiente porque usa mucho menos memoria y disco que una máquina virtual, esto hace que sea mucho más rápido ejecutar un contenedor con una máquina virtual, por lo que la aplicación estará levantada en menos tiempo, eso también es importante.



A diferencia de lo que es un Ward o un Zip, (en donde solamente se agrupan los binarios), el Container tiene todos los recursos necesarios para que la aplicación ejecute, por ejemplo, si estamos hablando de una aplicación generada para JAVA, contiene el runtime de Java, JDK, Tomcat, MySQL, variables de entorno, bibliotecas, binarios, etc.
Por eso decimos que un Container es una forma de virtualización que permite que una aplicación ejecute en un proceso aislado e independiente del resto, es como si fuera una caja donde está nuestra aplicación web, pero también el servidor Web, el administrador de base de datos, la librería, variables de entorno, etc. Es decir, todo lo necesario para que la aplicación ejecute dentro de esa caja.

El registry de nuestro computador, no tiene idea de lo que se está ejecutando allí, y nada que esté en el contenedor puede ver el Host (aunque esto último en realidad sí se puede configurar).
Por esa razón yo puedo tener inclusive distintos Containers con distintas distribuciones de Linux, porque se incluye en cada Container todo lo necesario para que la aplicación pueda ejecutar , ejecuta en un ambiente totalmente aislado porque no necesita de nada más para poder ejecutar, así que puedo tener un contenedor, por ejemplo, SQL Server, otro con Tomcat, Java, etc.

Todo este ambiente de ejecución, todo lo que se necesita, se define en lo que se llama una “imagen”, la que se declara en un archivo de texto generado por GeneXus, con lo cual le dice a Dockers todos los recursos que va a tener esa imagen.
Un contenedor entonces, es una instancia de una imagen y puedo crear tantos contenedores como necesite.
Acá hay una cierta analogía con el caso de orientación de objetos, en el cual la imagen podríamos pensar que es la clase y hacer una instancia a la clase, o sea, crear un objeto, en este caso el contenedor es una instancia de la imagen que estaba armada.



¿Cuáles son los principales beneficios que tiene ejecutar contenedores de Dockers (que la aplicación ejecute en contenedores)?

Para empezar, yo puedo ejecutar la aplicación nuestra en cualquier computador, así sea una Mac, o sea una computadora Linux, o Windows, todo ejecuta en un contenedor, es decir que lo que voy a tener que tener instalado es el Docker for Mac o el Docker for Linux, para que pueda crearse el contenedor allí, ese contenedor entonces lo voy a poder llevar a cualquier computadora y ejecutarlo.
Ya vimos que cada contenedor incluye todo lo necesario para que una aplicación ejecute.

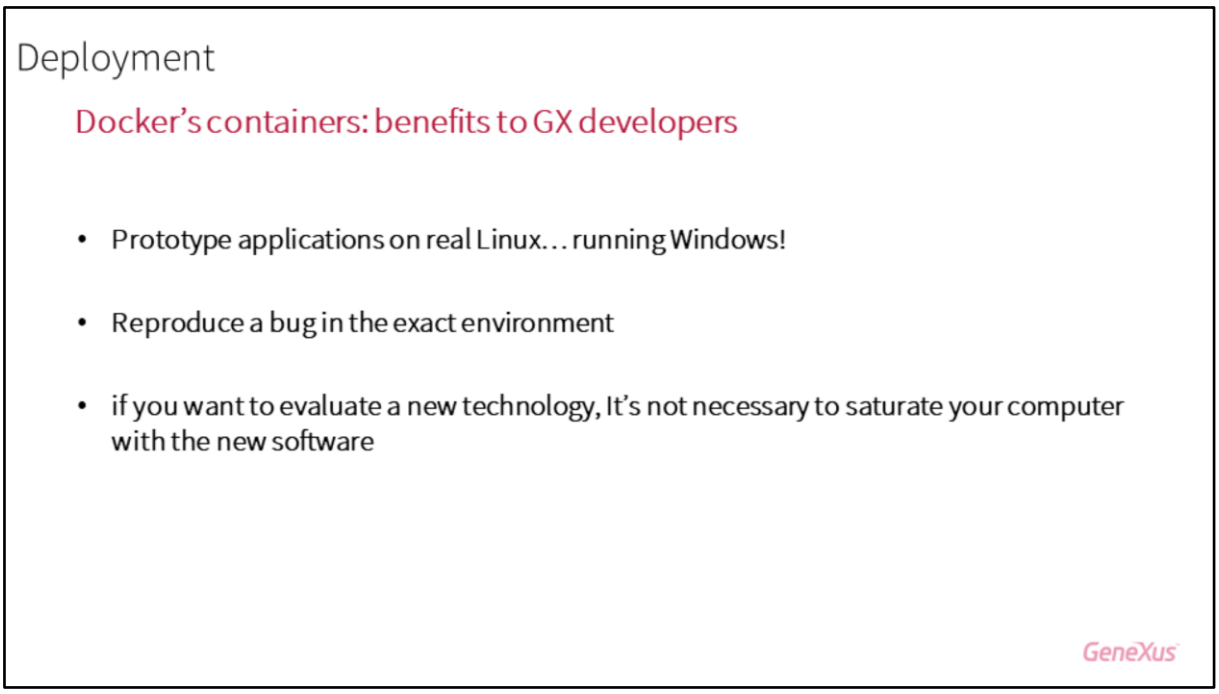
Ahora, si queremos varías instancias de nuestra aplicación ejecutando a la misma vez, ¿se puede tener a nuestra aplicación instalada en varios contenedores, cada uno con la base de datos?
Sí, se puede, como dijimos… un contenedor puede incluir todo lo necesario para la aplicación, incluyendo a la base de datos, pero justamente, la ventaja de usar contenedores es que puedas tener ejecutando varias instancias de la misma aplicación, que la base de datos sea externa a los contenedores y que sea excedida por todos ellos.
Si cada instancia de la aplicación tuviera su propia base de datos, los usuarios de la misma aplicación verían datos distintos según el nodo al que están accediendo.
La recomendación es que la base de datos esté afuera, podría incluso estar en un contenedor, aunque probablemente lo mejor es que no, sino que esté en un servidor físico con toda la potencia que pueda precisar, de esta manera, todos los nodos de la aplicación estarían accediendo a la misma base de datos y si mañana se necesita escalar la aplicación, lo único que se necesitaría hacer es instanciar más contenedores con idéntica configuración que accedan a la misma base de datos y si la carga baja, se pueden dar de baja contenedores.
Existen herramientas que proveen este tipo de orquestación, de que si los nodos superan, por ejemplo un 80% de carga, automáticamente se instancian más nodos.
Estas herramientas son estándar y trabajan independientemente de si la aplicación fue generada en GeneXus o con cualquier otra herramienta de programación.

Otra cosa para lo cual son útiles los contenedores, es porque si yo quiero probar una aplicación, en vez de por ejemplo instalarme un montón de cosas y haciendo que mi máquina se vaya degradando a medida que voy instalando cosas, porque evidentemente por más que instale y después borre, siempre el registry queda con un montón de basura, entonces, yo lo que puedo hacer es tener una imagen creada con el ambiente en el cual quiero instalar esas herramientas e instalarlas allí, en ese contenedor y a partir

de ahí me va a quedar funcionando la aplicación sin necesidad de que mi máquina física se entere de que estoy instalando cosas allí.

Otra cosa para lo que también es bueno los Dockers, es que cuando una aplicación está ejecutando dentro de un contenedor Docker y una aplicación, por ejemplo, tiene problemas porque está consumiendo demasiada memoria o porque puede quedar en loop infinito, o está consumiendo muchos recursos, esa instancia del contenedor no va a afectar otras instancias al contenedor que esté ejecutando, entonces no va a pasar como pasa hoy, en una máquina si yo tengo todo instalado en un mismo Web Server y en una aplicación que está demandando muchos recursos o mucha memoria, seguramente me va a afectar la performance de las otras aplicaciones, o la base de datos me va a afectar la latencia de las otras bases de datos, en cambio, si están ejecutando todos dentro de un solo contenedor, no voy a tener problemas de que una aplicación le genere problemas a otra aplicación.

Otra cosa también importante, es que es mucho más flexible lo que sería el ciclo de el desarrollo, el Test y el Deploy.
Si una cosa se prueba que funciona en desarrollo, uno puede hacer que en Testing se pruebe en exactamente las mismas condiciones, en producción puede haber alguna cosa que cambie, como por ejemplo los datos de producción son distintos a los datos de prueba, pero la prueba es en condiciones mucho más similares a las que tenemos hoy en día cuando tenemos que probar algo que pasamos de desarrollo a producción, o sea que la tecnología Docker en ese sentido es muy beneficiosa.

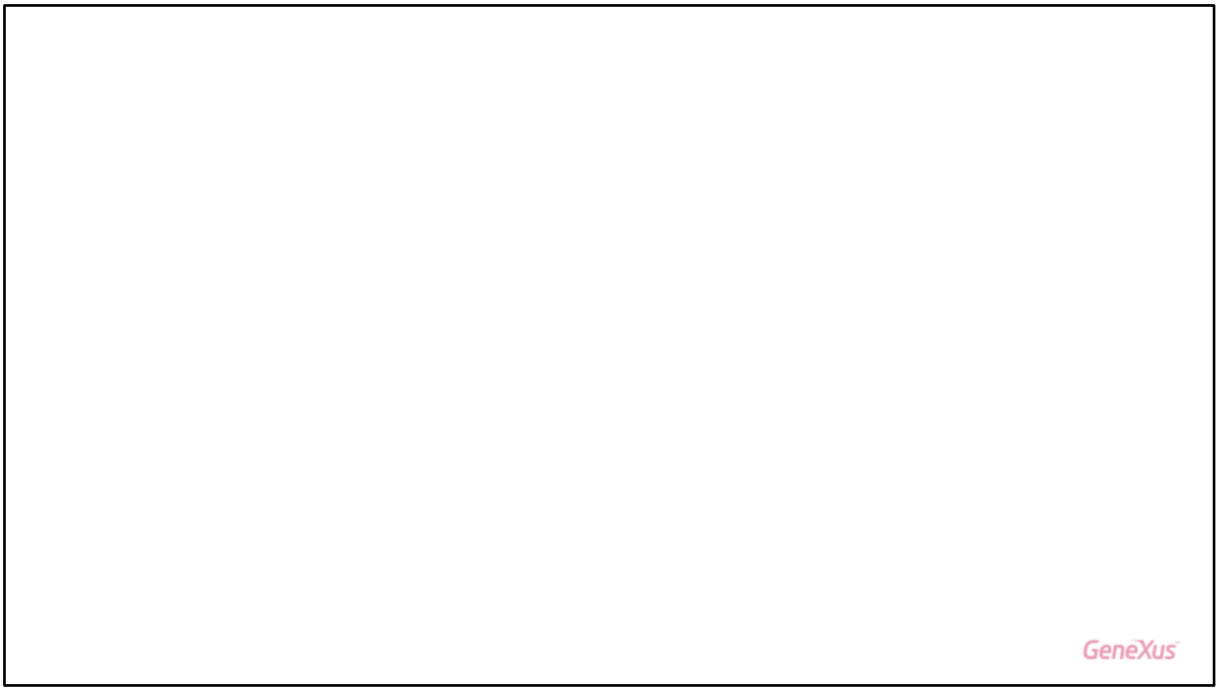


¿Que beneficios trae un poco mas concretos para los desarrolladores de GeneXus?

Es que es posible probar que una aplicación corra bien en Linux, porque el contenedor tiene todo el ambiente de Linux, aunque en realidad estemos ejecutando en Windows, es decir, que podemos probar una aplicación GeneXus Web en Linux, aunque nuestro sistema operativo en ese momento sea Windows. Y ejecutando en Linux en forma real.

También es posible probar un bug en idénticas condiciones a como lo encontró la persona de Testing, si quieren reproducir algo o el cliente tuvo un bug y quiere reproducir algo, lo puede reproducir en exactamente el mismo ambiente, utilizando para eso un contenedor.

Otra cosa más que puede ser útil es que si quiero probar una tecnología que no conozco y quiero instalarme cosas en la máquina puedo buscar si hay una imagen que me sirve e instalar el Software allí, así que son útiles los contenedores de Dockers para muchas cosas, ya que está corriendo todo en un ambiente aislado del resto de la máquina.



Antes de pasar, no se si hay alguna duda…
Había una cosa que por mientras les voy contando… algo muy útil que tiene esto, que capaz que pasó medio desapercibido, es que una vez que uno hace el Deployment en su ambiente de desarrollo, de test, esa misma imagen que se crea es la que se puede probar en Test y si anda bien eso mismo se pasa a producción, no hay que cambiar absolutamente nada en el medio.
Eso, aprovechando otra de las funcionalidades que mencionaba Rodolfo antes, que es el tema de las variables de entorno, seguramente mi aplicación vaya contra una base de datos de test y de producción obviamente contra otro.
Bueno, yo puedo levantar el mismo contenedor y decirle cuando lo levanto (vía variables de entorno) que vaya a la nueva base de datos, entonces es algo que hasta ahora no habíamos tenido o uno tenía que tocar el Web.Config cuando lo iba a pasar a producción o tocar distintos archivos de configuración, eso ya no es necesario con esta tecnología, junto con el otro cambio de variables de entorno que les decía recién.

Participantes:

- Con respecto al Web.Config, tenemos una aplicación que de repente en un F5 dejó de funcionar todo, web, mobile y después averiguando era porque el Web.config había superado los 250K y era una limitación del internet de information Server que después se arregló en el servidor con un cambio en la registry.
Dos preguntas, una es ¿es necesario todo lo que ponen en el Web.config? Me imagino que si, pero no lo entiendo mucho…
Después, ¿se puede partir ese archivo para que no supere esa cantidad de tamaño? Si tuviera algún problema, por ejemplo de subir al Docker, que uno pueda armar el mismo ambiente con Internet information Server, por ejemplo y si ahí también se me va a romper, ¿cómo se hace dentro de un Docker para tocar una registry para que ahora funcione de nuevo? Eso no lo entiendo muy bien…
  - Bien, la primer pregunta, si precisamos todo lo que está en el web.config, el web.config se genera muy grande en el ambiente local, en el F5, porque tiene absolutamente todo lo que tiene la KB. Cuando uno hace Deployment, por ejemplo, en el web.config están todos los endpoint de los servicios rest que tengas en la KB, pero capaz que en el Deploy de tu aplicación no van todos, van algunos, eso se saca toda esa parte y se agregan solamente las cosas que se necesita para ese Deployment en particular, entonces te diría que en el 90% de los casos, el web.config de un Deployment va a ser más chico que el web.config con el que trabaja GeneXus, que aparte tiene el Loguin prendido, tiene una

cantidad de facilidades para que te sea más fácil el prototipado justamente.
La otra pregunta, si se puede llevar a otro archivo… sí, nosotros no lo utilizamos, nunca había escuchado de este problema de que el web.conifg quedara demasiado grande… te diría que sería difícil que te pase en un ambiente después de hacer un Deploy, pero bueno, si llegara a pasar, habría que hacer esas cosas que hicieron ustedes.
Con respecto a cómo modificarlo dentro de un Docker, hay comandos específicos, pero ahí ya tenés que no ser un experto en Dockers, pero saber un poco, hay un comando que es Docker.exec que te sirve para entrar por línea de comando al "contenedor" cuando está corriendo y ahí por PowerShell o por alguna herramienta, todo este command line obviamente, podrías modificar la registry para "salir andando" y después de modificado, eso se puede "commitear" para que quede hecha una imagen con ya la modificación de la registry, entonces que no tengas que hacer lo mismo cada vez que levantes el container. Una vez que modificaste la registry, en tu caso, salvas eso como imagen y después usa eso para levantar en el resto y no vas a tener que hacerlo mas.

- Me imagino que yo creo que ya lo tienen que haber hecho, volviendo a la presentación anterior ¿con Jenkins yo después puedo activar y generar para lograr obtener el contenedor?
- Sí, porque esto va todo por tareas MSBuild, entonces se puede vía Jenkins programar una tarea MSBuild que haga el Deploy…
- Claro, pasó todo el Build correcto y genero mi contenedor.
- Si, exacto! Es más, como Docker es todo command line, podría incluso tener una tarea que ya lo publica a una registry, a la registry pubica de Dockers o de donde sea.

(Si no me equivoco hay una Demo en este videíto, de los contenedores.)

- Hicimos una prueba con Docker en .Net, y creó una imagen de 10G, ¿es eso normal?
- En el mundo Windows todavía sí, las imágenes son bastantes más grandes que en el mundo de Linux, para ser .NET, .Net Framework (el tradicional) corre solamente en imágenes Windows y esas son mucho más grandes.
  Si fuera tu caso, te recomendaría pasar a Net Core que sí corre en Linux, tenemos imágenes muchísimo más chicas y funcionan tanto en Linux como en Windows.
  Las alternativas son Net Core y Java-
- O sea, la alternativa para .Net es pasarse a Net Core o Java…
- Bueno, Microsoft está bajando los tamaños constantemente y agregando funcionalidades a Windows para poder justamente recortar, hace poco hicieron una muy grande para poder achicar justamente el tamaño de lo que sería una imagen Windows necesaria para correr una aplicación, capaz que llegan a los tamaños de Lniux, todavía están medio lejor.
- Gracias!

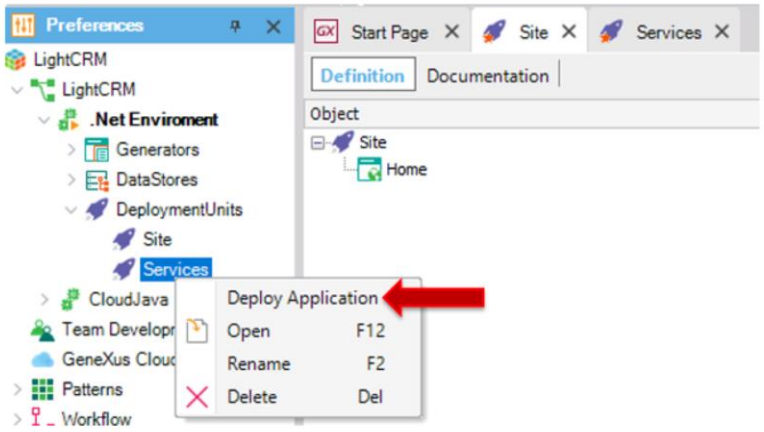
Deployment Unit Object

Vamos a ver ahora lo que sería el Deployment Unit Object

Esto es interesante porque muchas veces nos pedían que cuando estaba haciendo el Deployment de una aplicación, pudiera dejar guardado todo ese Deployment en algún lugar, cosa que después pueda repetir ese Deployment en otro momento.
Para eso se creó el objeto Deployment Unit

Deployment Unit Object

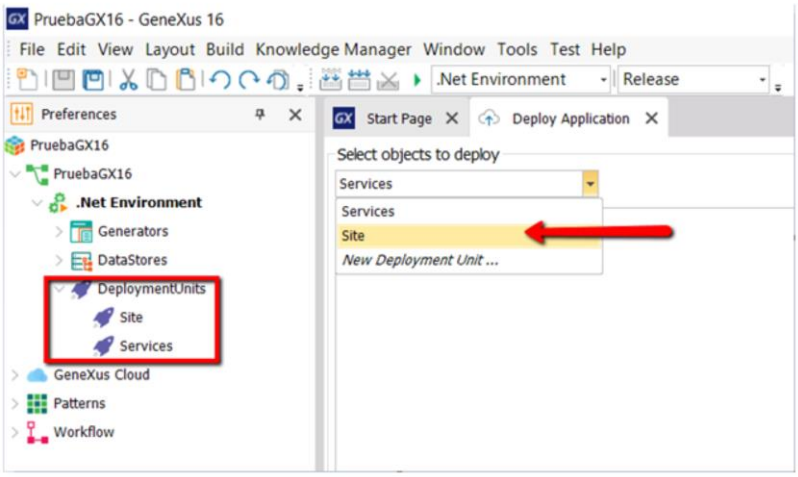
- Saves the set of objects to be deployed together, and we can deploy it later.

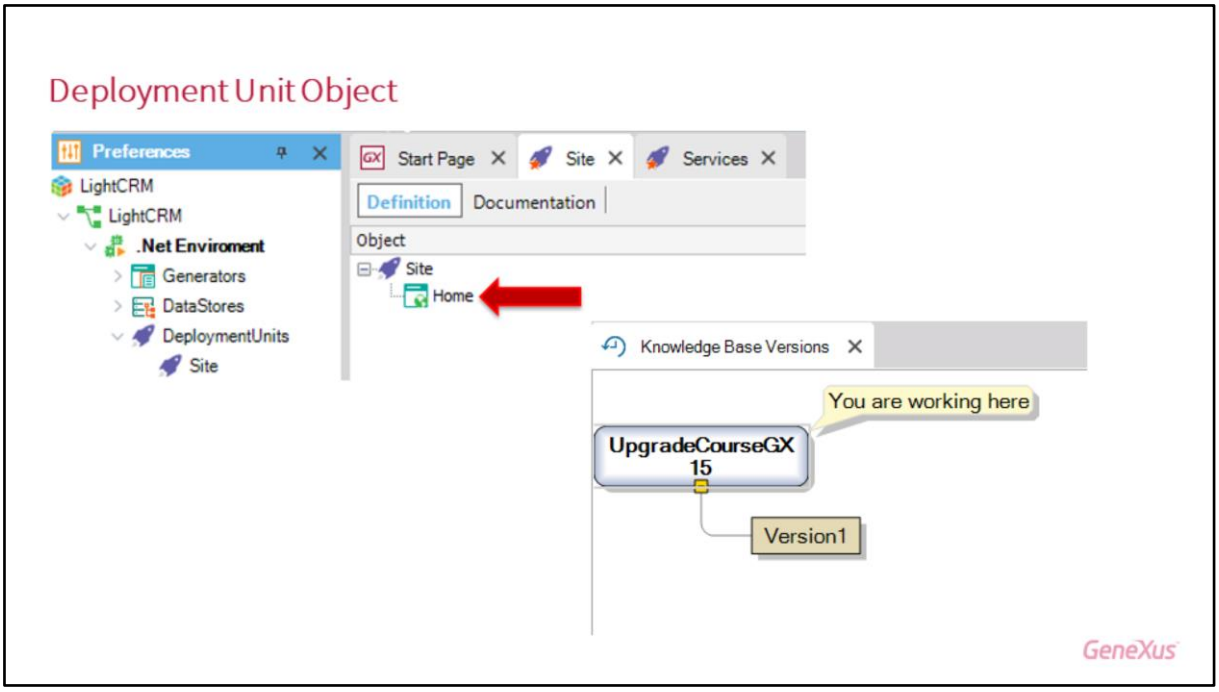


Bajo un enviroment van a ver que hay un nodo que se llama Deployment Unit y cuando yo abro la Application Deploymen Tool (Con Build Deploy Application) por defecto me va a crear una Deployment Unit, pero yo podría crearme distintas Deployment Unit, para hacer por ejemplo que mi aplicación se separe en partes y decir “bueno, en una Deployment Unit quiero poner solamente lo que es el sitio Web”, “en otra Deployment Unit quiero guardar el Deployment de los servicios Rest” y así sucesivamente, es decir que me permite guardar todos los objetos que formarían parte de ese Deployment y como puedo crear varias Deployment Unit, podría clasificar eso y hacer el Deployment por seperado de cada cosa, entonces cuando yo quiero le doy botón derecho, elijo Deploy Application y puedo hacer el Deployment solamente de esa Deployment Unit sin necesidad de hacer el Deployment de toda la aplicación completa, como en el caso de cuando habría Build Application y apretaba el botón Deploy.

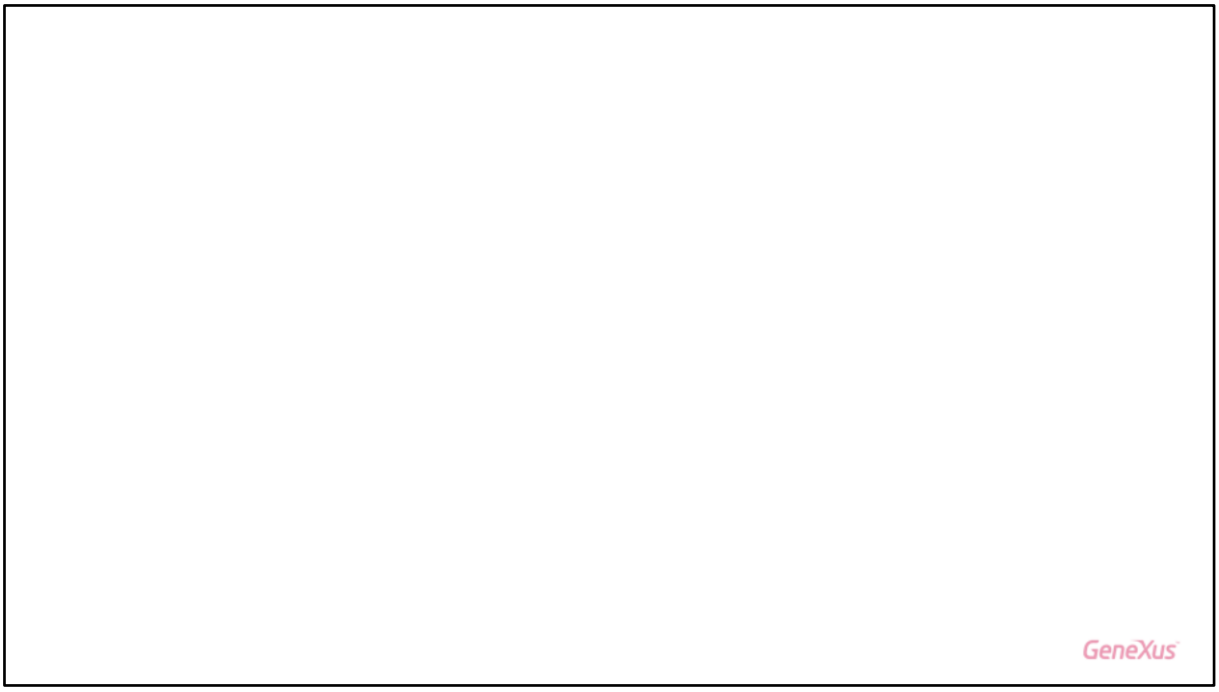
Deployment Unit Object

- Deployment of a specific Deployment Unit





Una cosa que podríamos preguntarnos, es si cuando arrastramos objetos a una Deployment Unit (en la ventana del Deploy Application) y luego modifico a esos objetos, si el Deployment me toma en cuenta los últimos cambios realizados a los objetos, o si al momento de arrastrarlos se congela el estado de los objetos y se hace el Deploy a como está el objeto al momento en que lo arrastré…
Acá tiene que quedar bien claro una cosa, como lo que estamos haciendo es un Deploy, en el cuadro del diálogo del Deploy Application quedan referencias a los objetos generados y compilados porque es lo que se va a llevar a producción.
Si se agregaron a la ventana objetos ya generados, pero después se cambia el objeto y no se genera, el Deploy va a usar lo que se había generado anteriormente.
Una cosa que debe quedar clara, es que si se crea una Frozen Version de la KB, estamos hablando de que se guarda el estado del objeto en cuanto a su edición, pero no se guarda el estado de lo generado o compilado del objeto, la forma de guardar lo generado o compilado es utilizando una Deployment Unit en la ventana de Deploy Application.



Veamos en GeneXus ahora estos conceptos que hemos visto, cómo hacer Deploy de una aplicación.

Aquí yo tengo creada una KB con una transacción, al cual le apliqué el Pattern Work With for Web y también mediante la propiedad Data se crea un Data Provider y puse datos de países, ejecuté la aplicación, aquí tengo el main (que es el objeto Home) y el Work With Country, estos son los datos que tengo programados en el Data Provider que se ingresaron en la base de datos.

Muy bien, yo quiero hacer Deployment de ésta aplicación, entonces para eso voy a Build, Deploy Application y aquí vemos la ventana para hacer el Deployment.
Tengo dos Environment, un Environment Java (Java Environment) y un Environment .Net (.Net Environment), si yo estoy en un Environment Java, para poder hacer Deployment de la aplicación , lo que tengo que hacer es arrastrar aquí el objeto main de mi aplicación, en mi caso, el objeto main es el Web Panel Home.
Y ahora lo que queda decir es elegir cual es el target…
Para poder hacer un Deployment a una imagen de Docker (que se cree una imagen de Docker a partir del cual se va a instanciar un Container, y en ese Container voy a poder ejecutar la aplicación con todos los recursos necesarios) voy a tener que tener instalado Docker for Windows, yo en este caso ya lo tengo instalado… y aquí tengo algunas propiedades que voy a poder setear, en mi caso no setee nada, lo dejé por defecto.
Ven que como estamos en un Environment para Java, si quisiera podría cambiar para que el contenedor fuera de tipo Windows, pero yo quiero hacer un contenedor Java.
En el momento que elegí Docker Image, tenemos algunas propiedades, por ejemplo, va a utilizar un Tomcat.8 (ya trae el nombre de quien va a mantener eso, el nombre de dónde va a estar la Web App dentro de ese Tomcat, el nombre de la imagen que es el nombre de la KB + el nombre del Environment (cursoactualizacióngx15javaenvironment)…

(Antes de dejarlo seguir a Rodolfo les quiero contar un poco más porqué vienen estas propiedades, en realidad porque nosotros sugerimos… no se si se lee bien, pero acá dice: “Tomcat:8.5-jre8-alpine”, esa es la imagen que usamos para empaquetar la aplicación adentro del contenedor nuestro, ustedes podrían cambiarla y poner otra imagen para otro, incluso para Application Server, entonces en ese caso también tienen que poner dónde es, que adentro del contenedor se va a copiar el War resultante, en este caso es “user/local/tomcat/webapps/”, si ustedes pusieran un Webphere o lo que sea, obviamente ese path va a cambiar.

Y lo otro que quería aclarar, es que los pasos de Deploy, la generación del War en este caso, son siempre las mismas, independiente de cuál es el target final. Y después hay un paso posterior que es qué se hace con ese target, en este caso, crear una imagen de Docker para que quede un contenedor.)

… Y ahora lo que vamos a hacer, es hacer un Deploy:
Al hacer esto, lo que va a hacer GeneXus es descargarse todo lo necesario del sitio de Dockers, entonces en particular va a ejecutar ciertos comandos de Docker y va a obtener lo necesario, por ejemplo, en este caso está obteniendo el Tomcat, descargándoselo, y todo lo que se necesita para ejecutar esta aplicación Java, esperemos unos segundos…
Muy bien, aquí vemos que el Deployment se hizo con éxito, quiere decir que hicimos un Deployment a una imagen de Docker, para poder ver las imágenes de Docker tenemos que abrir una ventana de comando, así que voy a ejecutar una ventana de comando y voy a dar el comando que habíamos visto para poder ver las imágenes de Docker, voy a utilizar el comando "docker images" y aquí vemos que me creó dos imágenes, una que es la que me dijo que me iba a crear (cursoactualizaciongx15javaenvironment), aquí tenemos a la derecha el nombre que le iba a asignar a la imagen y también crea una imagen para el Tomcat (8.5-jre8-alpine), ésta es la imagen creada, pero para poder ejecutar la aplicación lo que tengo que hacer es que a partir de esta imagen se instancie un container…

(perdón, quiero mostrarles una cosa, acá se ve lo que dice Rodolfo, dos imágenes de las cuales una es, la de Tomcat (la que dice acá arriba) y otra es la que GeneXus creó, si se fijan, una pesa 106 MB y la otra 114 MB, no tengo 220 MB en uso en mi disco en ese momento, porque la imagen nuestra son los 106 MB + la diferencia para llegar a los 114 MB, en el disco yo no estoy usando 220 MB, Docker es lo suficientemente inteligente para saber que la imagen generada por nosotros es la de Tomcat + otras cosas que nosotros le agregamos)

Participantes:

- Con respecto a la versión de Java ¿desde cuando está siendo compatible para trabajar con Dockers en éste caso?
- GeneXus 15, Upgrade 8
- Ahora, cuando tenemos ahí nosotros Tomcat, ¿me está diciendo que Tomcat es un Docker que contiene Tomcat?
- Sí, ya tiene el Tomcat adentro, el 8.5 en este caso…
- ¿Ese Tomcat, yo lo traigo o lo consigo ya pre-definido o ustedes lo tienen mas o menos armado?
- No, bueno, es eso que dice acá arriba, nosotros estamos usando una imagen oficial de Tomcat que ellos dan para que la gente que quiera hacer Deploy de sus aplicaciones en contenedores, puedan basarse en sus imágenes oficiales, ellos tienen esa imagen y nosotros lo que hacemos es bajar esa imagen y agregarle las cosas," la aplicación digamos" arriba.
- Perfecto! ¿y en el caso de .NET cuando trabajamos con internet information server?
- También tenemos una imagen…
- ¿y el tema del licenciamiento ahí como anda?
- Eso no tiene, no te cobran eso, porque el Windows que va ahí es Windows Core o Nano Server, alguno de esos dos, (nunca me acuerdo cuál es) que no tiene un tema de licenciamiento el armar la imagen con eso, capaz que después un proveedor te puede cobrar el levantarlo, pero lo puedes levantar tu en tus propios servidores
- Ok, gracias!

… entonces, para poder crear un container uso el comando "docker run --rm" para que cuando termine la ejecución se borre automáticamente, "-p" para decirle el puerto y yo quiero que se ejecute en el puerto "9999" o sea que el puerto ":8080" del Tomcat se va a mapear al puerto "9999" y la imagen que quiero a partir de la cual se cree el container es la siguiente que dice aquí arriba "cursoactualizaciongx15javaenvironment" (me quedó medio largo el nombre de la imagen)…
Ahí está instanciando entonces la imagen.

Voy a abrir otra ventana de comando y en esta otra ventana de comando voy a ver si efectivamente quedó instanciado el container, entonces para eso le doy comando “docker ps” y aquí veo que está creado este contenedor, que es donde va a poder ejecutar mi aplicación.
Entonces, la aplicación va a ejecutar en el puerto “9999”, lo que vamos a hacer ahora es escribir aquí cuál va a ser la URL de la aplicación, sería: “//localhost:9999/CursoAct”, este es el nombre que yo le di a GeneXus en la aplicación, si vamos a ver en GeneXus cuando creé le puse “Curso Act”, entonces va a ser esto, “/servlet” y después el nombre correspondiente al paquete Java, que eso lo tenemos aquí en GeneXus (el Java package name), “/com.cursoactualizaciongx15.” y finalmente el nombre del objeto que quiero que se ejecute, que es el objeto “home”.
(//localhost:9999/CursoAct/servlet/com.cursoactualizaciongx15.home)

Muy bien, aquí tenemos la aplicación corriendo dentro del containar desde el puerto “9999”…

(Un comentario, que esto se hizo antes de que saliera la 16, ahora va en el root de la aplicación, de todo lo que puso ahí Rodolfo, después del “9999” del puerto, el nombre de la aplicación no va, queda en el root del contenedor)

… y aquí tenemos a los datos ya en la base de datos, que está corriendo adentro de ese container con ese Tomcat en esa dirección que vimos.
Una cosa que podemos ver, es si efectivamente esto está corriendo en Linux, porque estoy ejecutando un Tomcat que está instalado en un Docker, que está interaccionando con un sistema operativo Linux, bueno, para probar que efectivamente estoy ejecutando en un ambiente Linux, para mostrarle que estoy ejecutando en eso, lo que voy a hacer es tratar de ejecutar el Bash, entonces para eso voy a hacer un comando “docker.exec -i -t” voy a hacer una referencia al Docker que está creado, es el Docker 9e0 /bin/ bash.
Vamos a ver si estamos en Linux, hago un “ls”, vamos a hacer un “CLS”, no, claro porque CLS es un comando de DOS, “Clear” es el comando para limpiar la pantalla en Linux.
Y bueno, yo acá podría moverme por ejemplo a la carpeta “bin”, ver el sub contenido, etc. Es decir que estoy efectivamente dentro de un Linux.
Vamos a salir (exit)
Y ahora ya tenemos una instancia corriendo en el puerto “9999”, lo que podría hacer yo es crear otra instancia en otro puerto distinto, entonces podría ser crear otro container “docker run –rm -p”, pero ahora lo voy a hacer en el puerto”9991”… y lo que voy a hacer es abrir otra ventana, pero ahora esta instancia debería estar corriendo en este puerto…
Y aquí tenemos uno en el “9999” y otro en el “9991”, pero los dos están apuntando a la misma base de datos, quiere decir que si yo voy acá (Upgrade) y cambio algo de esta instancia, voy a la otra, hago un refresh y ven que cambió, es decir que en realidad tengo dos aplicaciones ejecutándose a la misma vez en dos contenedores diferentes, pero están accediendo a la misma base de datos.

Para poder entonces terminar con esto, voy a ver cuáles son los contenedores que tengo corriendo (ven que tengo los dos contenedores, el que creé primero y el que creé después) y lo que voy a hacer es simplemente matar estos contenedores, porque no quiero seguir utilizándolos, “kill 3d9”
Ahora tengo uno solo y voy a hacer un “kill” entonces del segundo, “kill 9e0” y ahora no tengo ningún contenedor, pero sí sigo teniendo la imagen del curso de actualización y la imagen del Tomcat para poder instanciar los nuevos contenedores.

Vamos a cerrar entonces esta ventana de comando y ya que estamos ahora en GeneXus, vamos a ver otro concepto más que habíamos mencionado, las Deployment Unit:
Ven que en el environment tengo ahora aquí una Deployment unit que por defecto se llama “DeploymentUnit1” y que se creó automáticamente cuando hice el Deploy, pero por supuesto yo a esto le puedo cambiar el nombre, por ejemplo, si yo quisiera decir que “quiero separar por distintas unidades”, los distintos tipos de partes que tiene mi aplicación, entonces aquí solamente voy a poner lo que sería el “site” o el sitio web y quisiera crear otro Deployment unit que se llame “Services”, donde ahí podría incluir todos

aquellos servicios rest que yo tenga en mi aplicación, que quiero que estén dentro de una “Deployment Unit de Services” cosa que cuando quiero, puedo hacer el Deployment del sitio web o el Deployment de los services en forma independiente.
Por ejemplo si yo vengo aquí y le doy botón derecho, donde dice “sitioweb” puedo directamente hacer Deployment de la Application, con lo cual me abre de vuelta esta ventana, ven que ahora aquí tengo las Deployment Unit.
Esto en definitiva lo que me permitió es guardar, si yo por ejemplo voy a Local, voy a poder guardar todo esto asignado a una cierta Deployment unit y esa Deployment Unit la puede hacer el Deployment en el momento que yo quiera, o sea que yo puedo “salvar”, armar toda una estructura para hacer un Deploy, pero no armar la estructura y hacer Deploy en el momento como hicimos hoy, sino hacer la estructura y que queden guardadas en esta Deployment unit y después con botón derecho hacer Deploy en ese momento que a mi me interese y hacer Deploy por un lado, de por ejemplo, el sitio web, los servicios u otras partes en las cuales yo quiera dividir mi aplicación.

Bueno, ¿alguna otra pregunta?

- Dos preguntas, una ¿con los Deployment tienen algo del estilo de lo que se hizo en el Build, de tener comandos pre-Build y post-Build para el Deploy también?
- No, para eso no tenemos en GeneXus, pero sí en los scripts de MSBuild que nosotros tenemos para hacer los Deploy de esto, tenemos puntos de extensibilidad, hay tareas MSBuild donde podrías engancharte y ejecutar cosas antes y después o después de cada Deployment, eso sí tenemos…
- Ok, ¿está documentado en la Wiki eso?
- Está documentado en la Wiki…
- Y la otra pregunta, era que nosotros venimos usando Deployment Unit pero las properties seguían siendo las mismas cuando te cambias de Deployment Units, y eso no se si lo tenían previsto para incorporar…
- Sí, estamos trabajando en eso, en que queden las propiedades también salvadas en el Deployment Unit.
- Gracias!

Package/Release/Deploy

Deployment Unit Object
Build events
Serverless (deploy to AWS lambda)
GX Cloud Deployment Services → F6 PAAS
Containers (Docker)

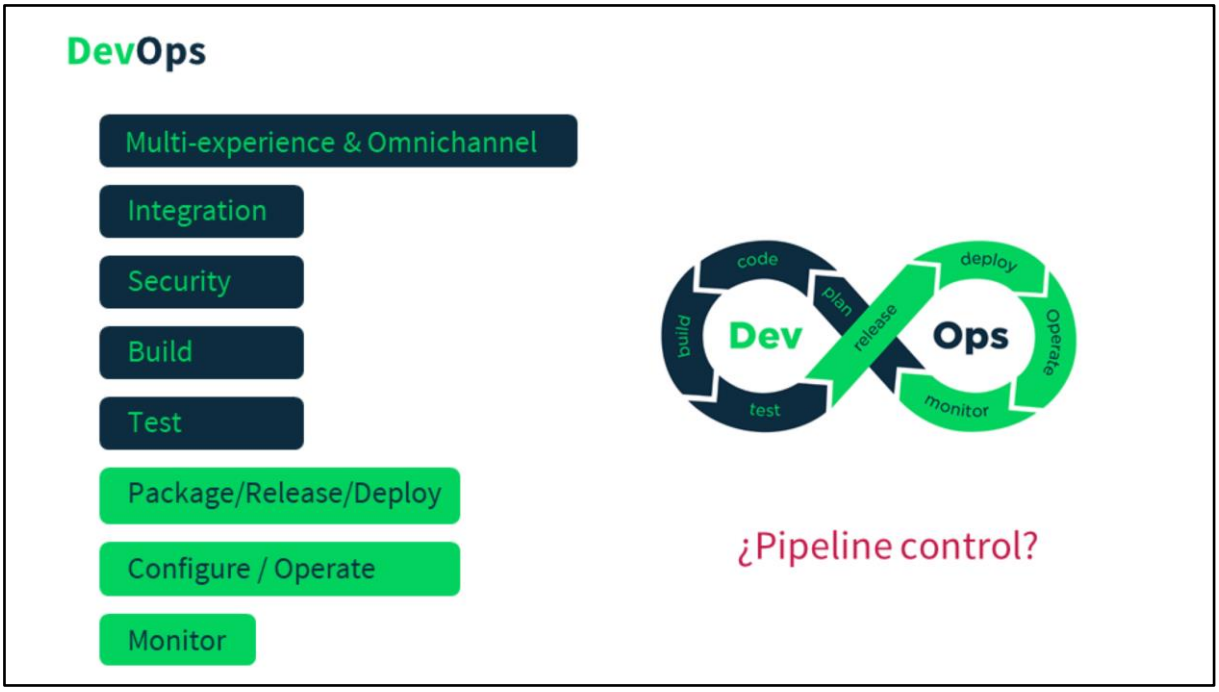
Configure/ Operate

ConfigurationManager API
Config via Environment variables

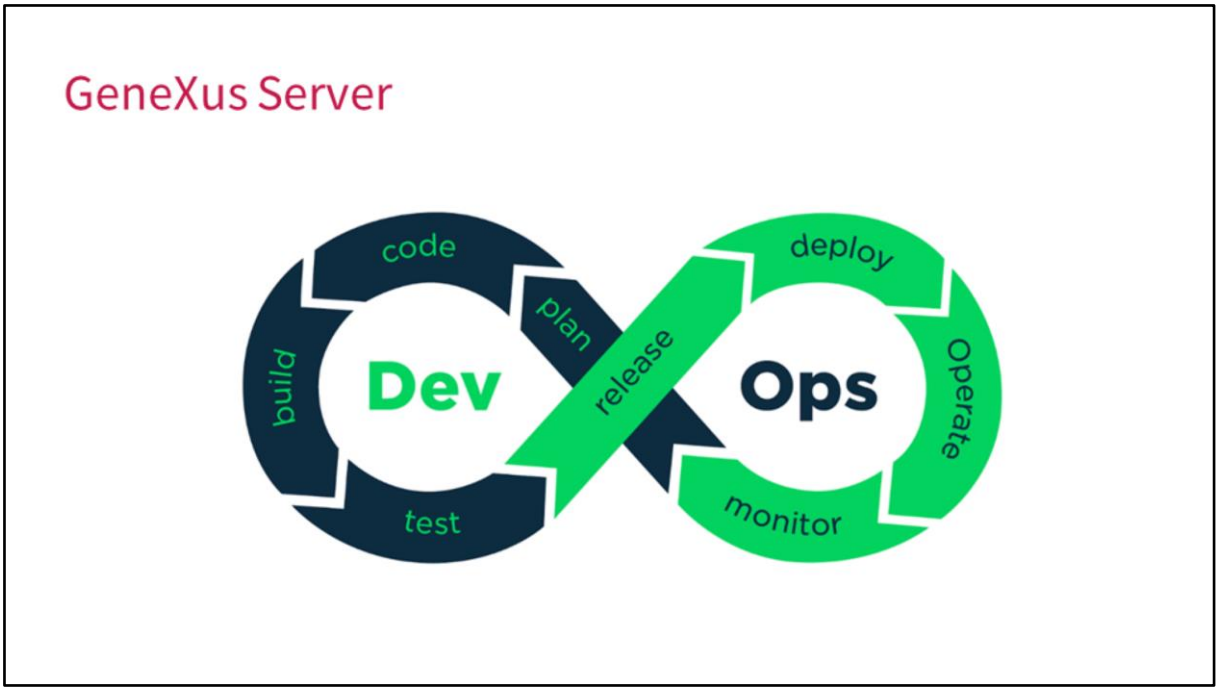
Monitor

Log API

DevOps



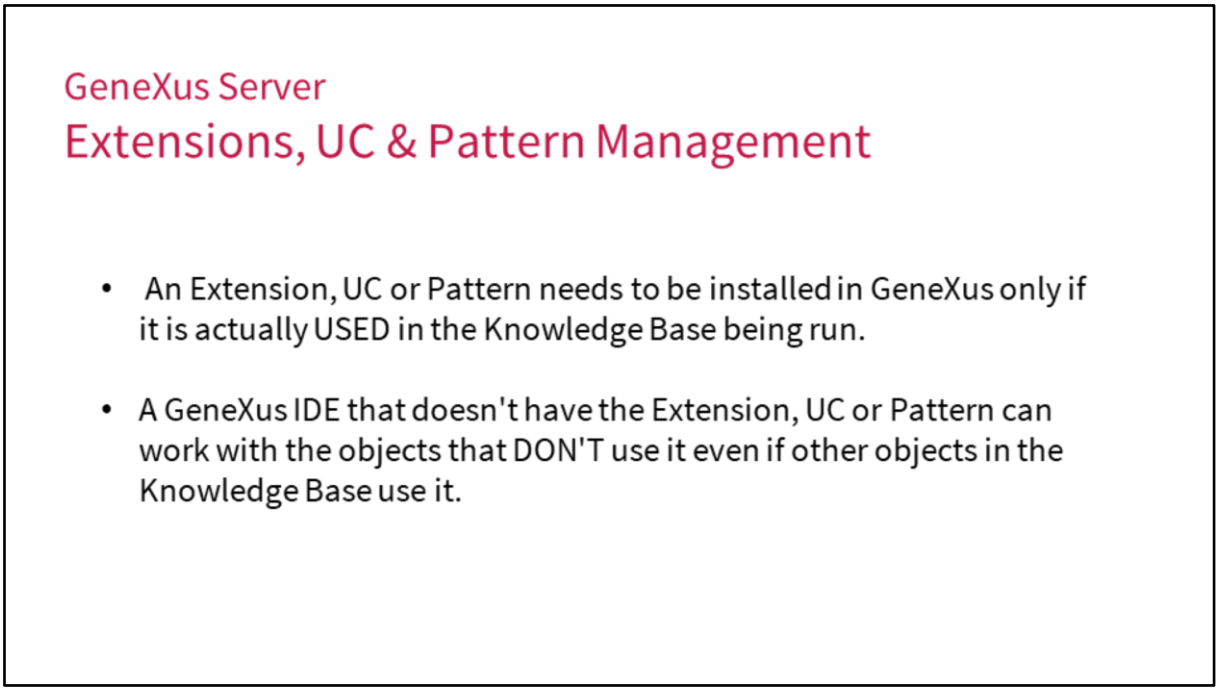
¿Cómo hacemos para facilitar el flujo de éste ciclo?



Primero que nada, vamos a ver ahora algunas novedades en el GeneXus Server que es una pieza clave cuando trabajamos en equipo, para poder integrar el código de forma sencilla.

En esta versión, hay mejoras que se orientan a simplificar el tema del manejo y mantenimiento de las instalaciones del GeneXus Server.

Desde el Upgrade 2 de la GeneXus 15, es posible actualizar una instalación de una instancia de GeneXus Server, diferente a la instancia por defecto.

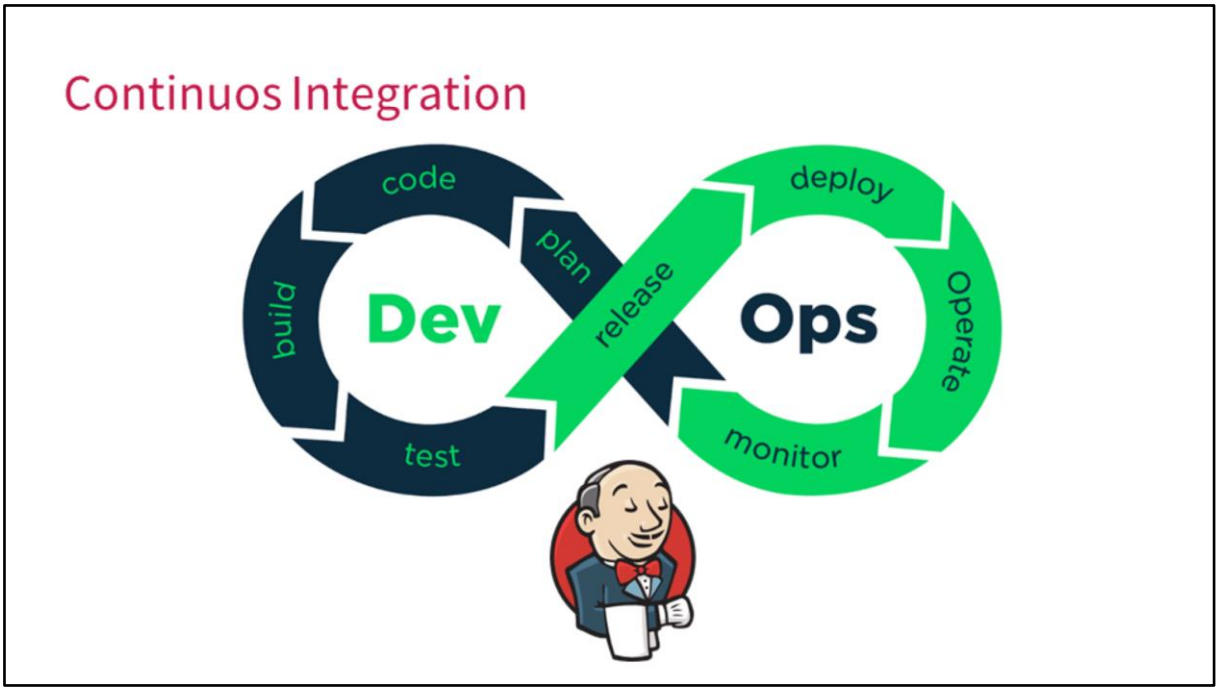


Por otro lado, hay una mejora importante en el manejo de las extensiones “User Controls” y “Patrones”, para simplificar el tema de la instalación de los mismos en el propio Server y en todos los GeneXus IDE que interactúan con el Server.

Cuando trabajamos en una KB que utiliza alguna extensión, algún patrón o User Control, al subir esa KB al Server, necesitamos que esa extensión también esté disponible en el Server, esto es porque las extensiones definen muchas veces partes nuevas de objetos existentes y para que ese objeto pueda subir al server con esas partes nuevas, el server también necesita conocerlas.

Ahora bien, antes si yo tenía en mi instalación de GeneXus una instalación que no se usaba en ningún objeto de la KB y subía mi KB al Server, necesitaba instalar la extensión también en el server.
A la misma vez, cualquier instancia en GeneXus que bajara esa KB del Server necesitaba también tener la extensión, por tanto, si en una KB se utilizaba la extensión, era necesario que tanto el Server como todos los GeneXus conectados, alguna KB alguna vez Hosteada por ese Server tuvieran instalada la extensión.
Desde la 15, Upgrade 3, las partes definidas por las extensiones se marcan como opcionales automáticamente y son enviadas al Server solamente cuando es estrictamente necesario, es decir, solo se necesita tener una extensión instalada en GeneXus si efectivamente la misma es utilizada en la base de conocimientos con la que se está trabajando.
Si una extensión está instalada en GeneXus Server porque alguna KB de las que almacena la utiliza, puede tener otras KBs que no la utilicen y no se agregan referencias a la misma, de esa manera, cualquier GeneXus IDE que se conecta a las KB que no necesitan la extensión, no precisan instalarla.
Y un IDE de GeneXus que no tenga la extensión, también puede trabajar con todos los objetos que no utilizan esa extensión, por mas que los otros objetos de la KB sí la necesitan.
De esta manera se simplifica bastante el hecho de poder tener un Server con KB que utilizan diferentes patrones, extensiones o User Constrols y no es necesario que todas las KB o todos los GeneXus IDE que se conectan al Server compartan las mismas instalaciones.

¿Está claro esto?



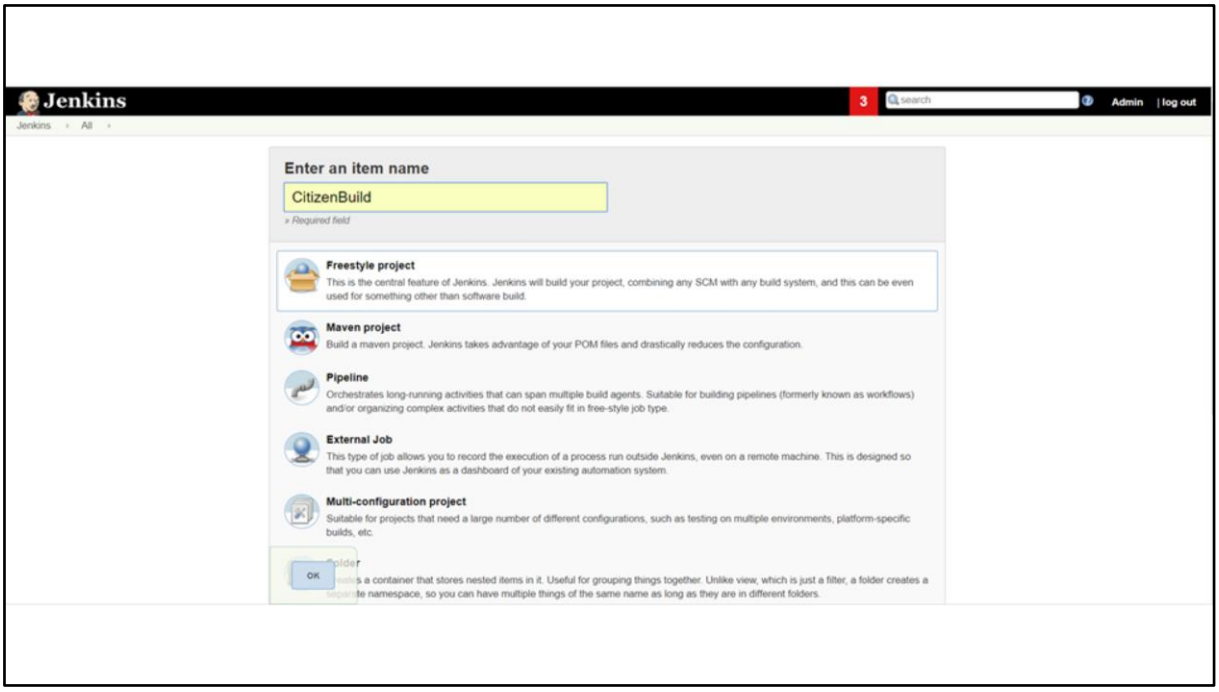
Hablamos del GeneXus Server, que es una herramienta fundamental para simplificar la integración del código, pero realmente para poder alcanzar un flujo de DevOps, necesitamos comenzar implementando una integración continua.

Una novedad en esta versión, es que ahora GeneXus también se integra a Jenkins como motor de integración continua.

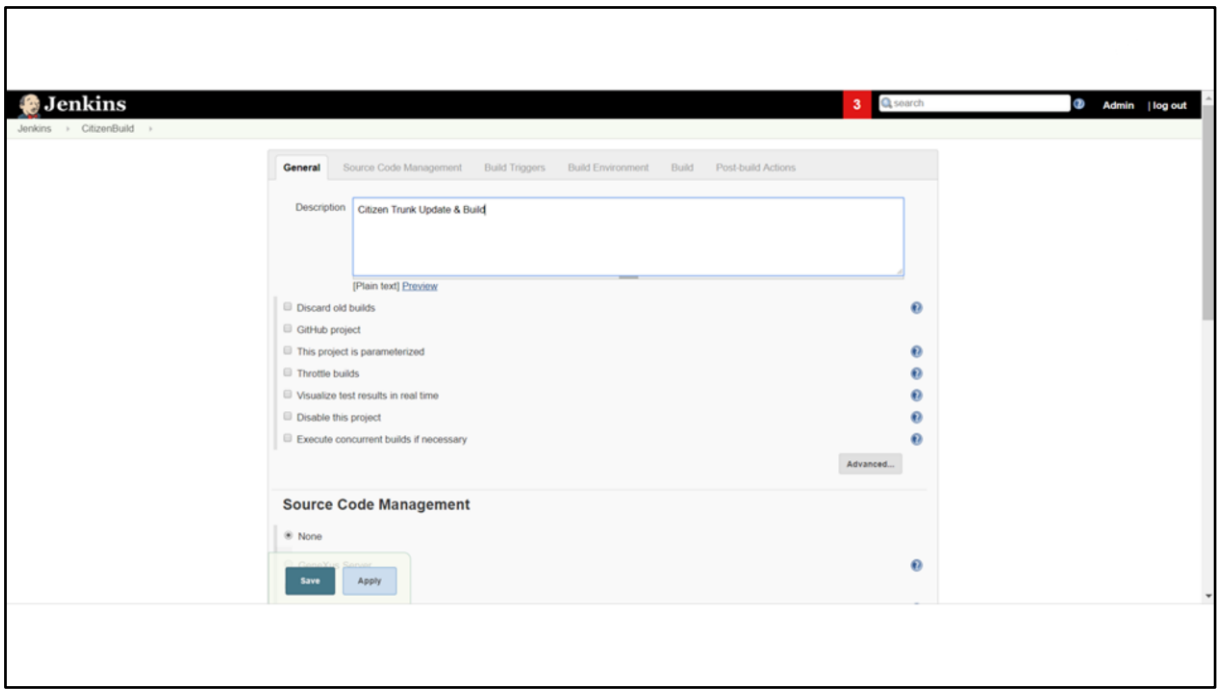
Vamos a ver cómo automatizar un caso básico, que sería cuando hay un cambio en el código que se dispare automáticamente un Build.

Para esto vamos a ir Jenkins, vamos a hacer click en la opción Item y esto nos va a permitir definir un nuevo proyecto en Jenkins.

(Por las dudas, muchas de las cosas que vamos a ver acá en las pantallas hay que instalarle plugins a Jenkins para que tenga toda esta funcionalidad, pero está todo documentado en el Wiki y dice exactamente qué es lo que hay que hacer y cómo)

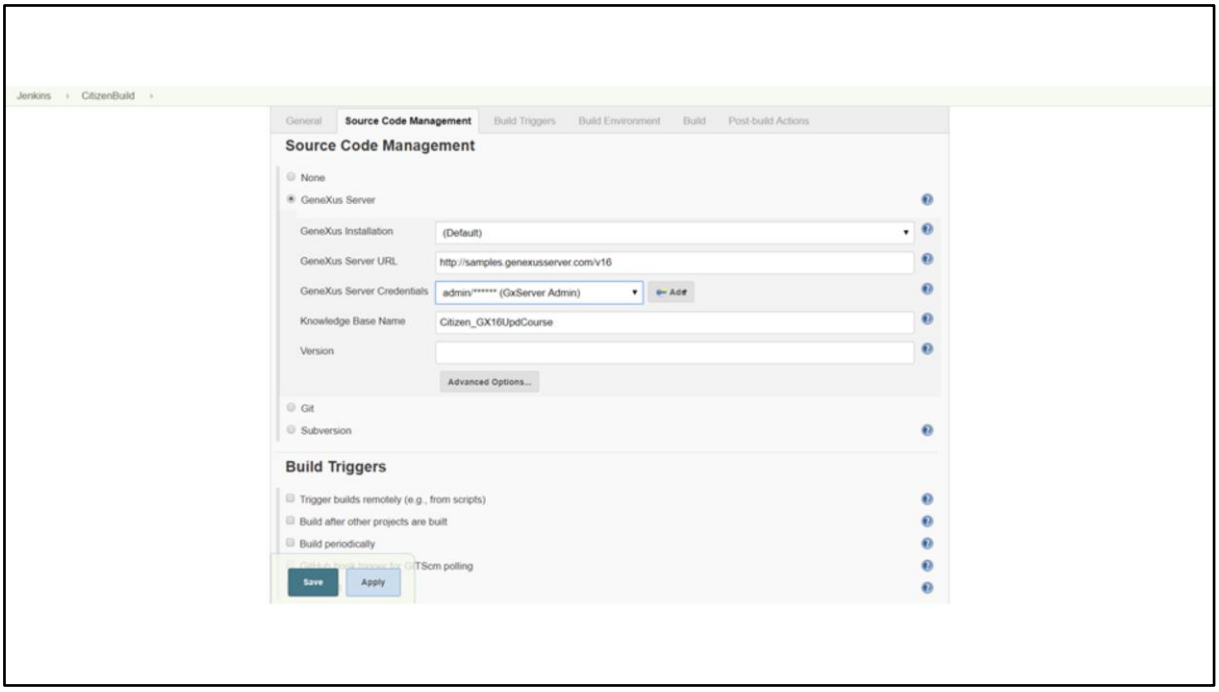


Al definir un nuevo proyecto, le damos un nombre, por ejemplo “CitizenBuild” y elegimos la opción Freestytle Project, luego hacemos click en el botón “ok"



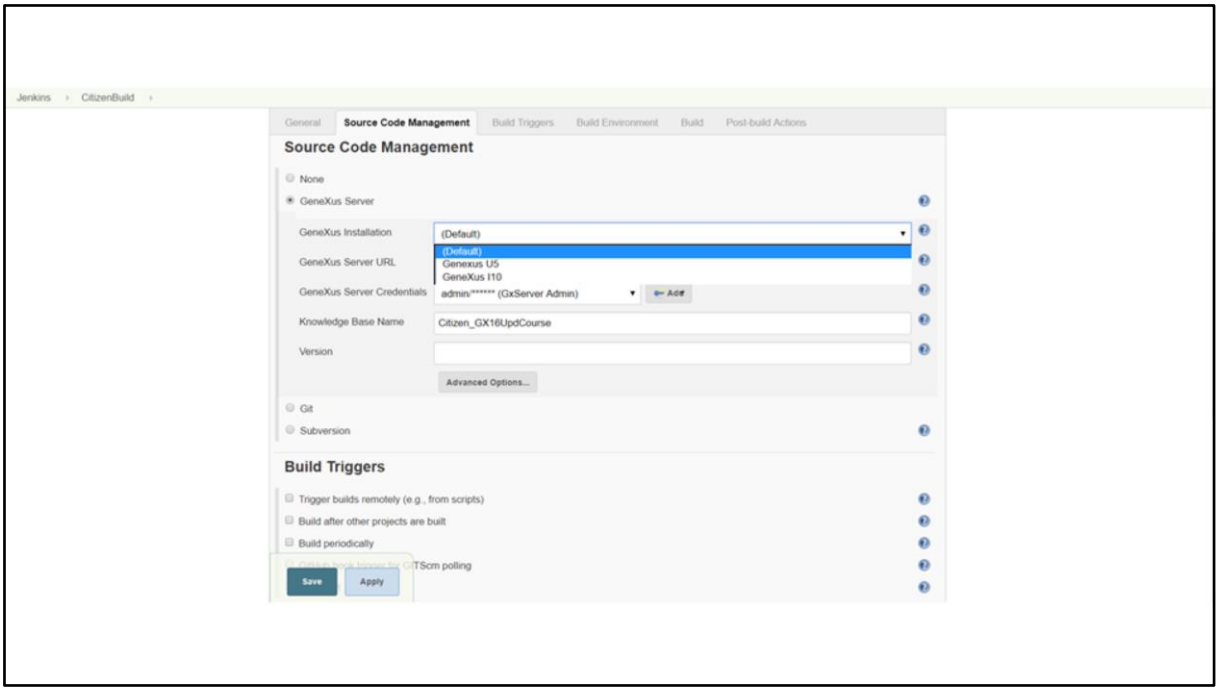
Así vamos a editar las propiedades de nuestro poryecto.

Le vamos a poner al a descripción que estamos haciendo un Update y un Build de la KB del Trunk o la KB de desarrollo.

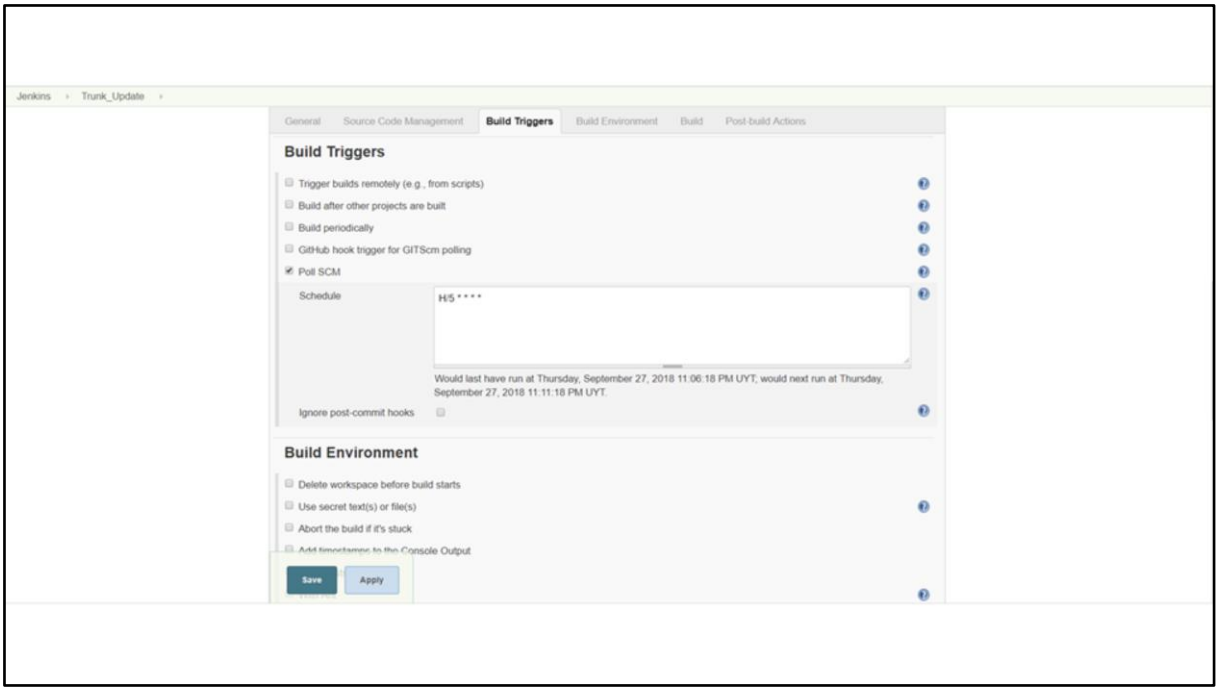


Si vamos a la sección de Source code Management, vamos a elegir GeneXus Server como nuestro repositorio, vamos a ingresar la URL de GeneXus Server, las credenciales que podemos definir varias, pero esto serían las credenciales con las cuales nos vamos a conectar a ese Server y el nombre de la KB a la que nos queremos conectar.

Si dejamos todo lo demás por defecto, estamos básicamente trabajando con la versión por defecto, es decir el Trunk y con el ambiente por defecto.

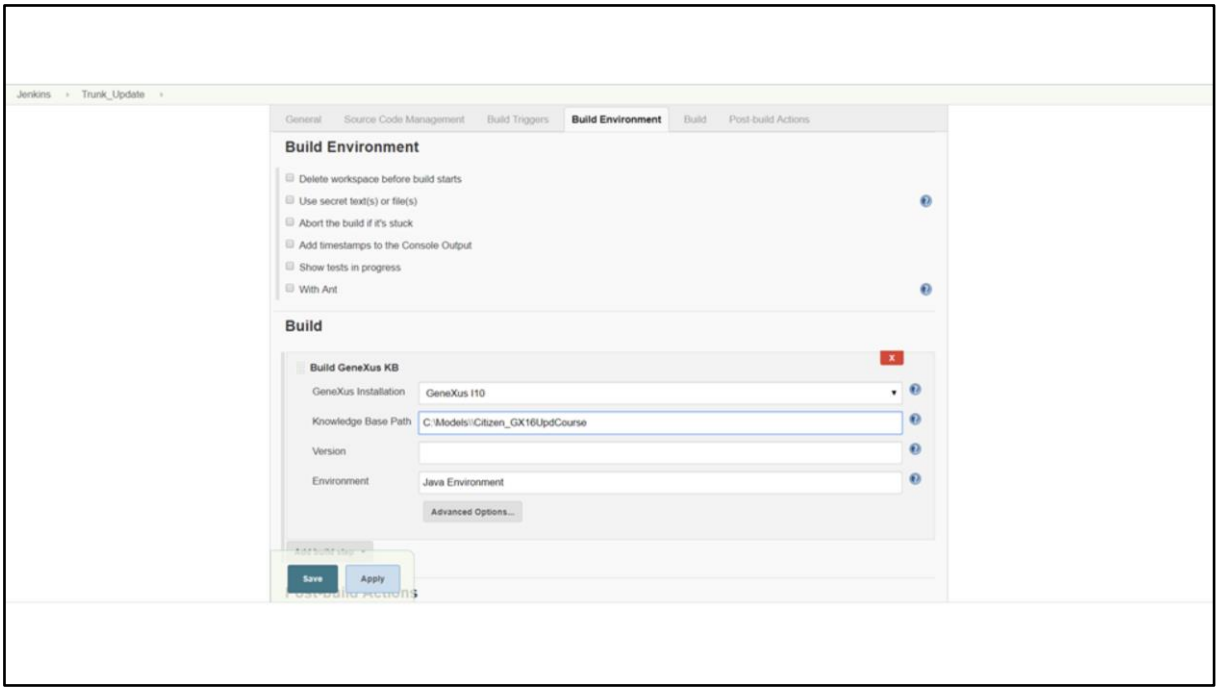


Podemos tener varias instalaciones de GeneXus y las definiríamos todas en la configuración de Jenkins.



Luego vamos a la sección de Build Triggers, que eso nos permite definir cuándo queremos que se ejecute ésta tarea o éste proyecto en Jenkins.

Aquí la opción que podemos elegir es “Poll SCM” es decir, le estamos diciendo a Jenkins que esté monitoreando el repositorio del código, en este caso cada 5 minutos y que en caso de que encuentre una nueva modificación, es decir un nuevo commit, dispare el proceso de Build.



En el sector de Build vamos a elegir la instalación de GeneXus y el Folder en donde se encuentra la KB. Nótese que la instalación del Jenkins está en la misma máquina en donde tenemos la instalación de GeneXus que hace el Build.

(Eso es solo aclarar porqué ven esos dos puntos, no tiene porqué ser así, a partir de un Upgrade que se hizo hace poco)

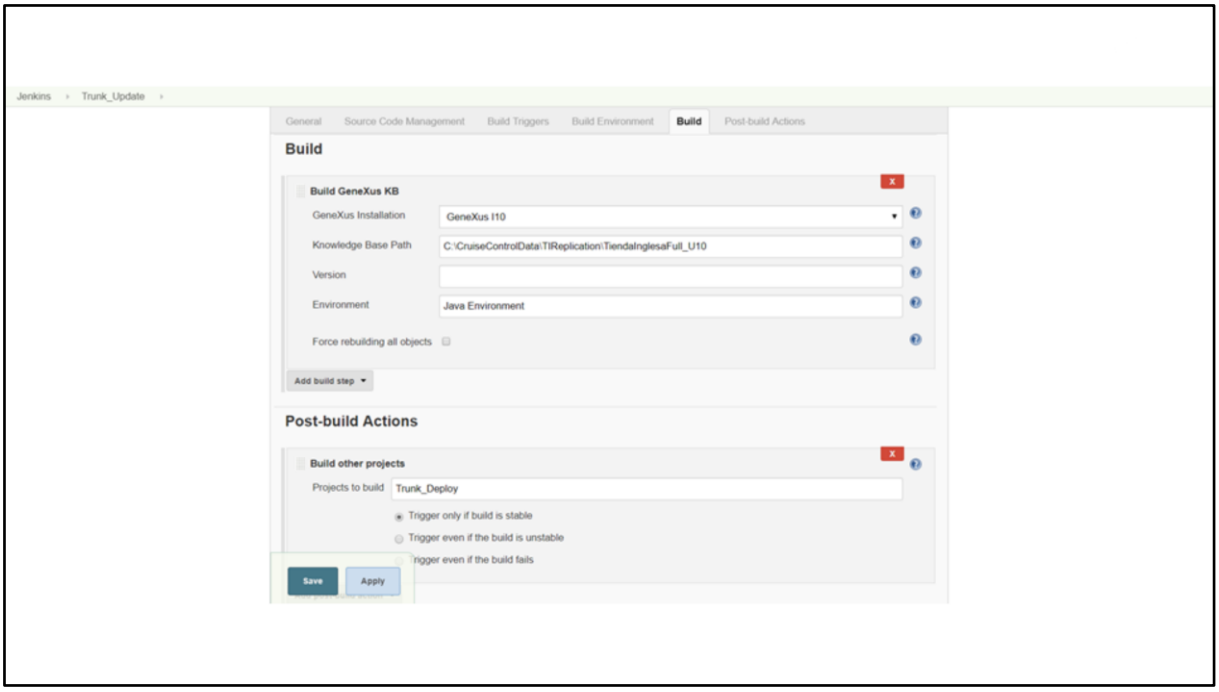
Participantes:

- Ahí se habla de toda la Kb, pero supongamos que tengo una KB corporativa y tengo algunas cosas ahí, quiero subir la parte financiera solamente…
- ¿Un módulo decís?
- Un módulo, exactamente…
- No, acá vos podés conectarte a una KB, dentro de la KB a una versión y a un ambiente, pero módulos no… o sea, es toda la KB.
- ¿Y va a estar previsto en algún momento que se haga por módulos o algo así?
- Sí, esa partición en realidad corresponde hacerla en la propia KB, es decir, tu podrías ese módulo hacerlo en una KB separada, exportarlo como interfaz y binarios y en otra KB que lo utiliza referenciar a ese módulo de tal manera que reciba los binarios y lo maneje solamente por la interfaz, pero esto lo que está haciendo es automatizar un Build de una KB, es decir, la parte de modularizar y separar en diferentes componentes en la KB, lo tienes que hacer en la parte de la KB, no te va a hacer esto, esto automatiza a partir de una KB, sí se puede elegir dentro de una KB si tienes muchas versiones y dentro de cada una de esas versiones, eventualmente muchos environment, tu puedes definir sí que quieres que te maneje solamente tal versión, tal environment, ella no lo mostró ahí, decía “cuando está todo por default”, pero ahí hay un botoncito que dice “Advanced Options” (opciones avanzadas) que es justamente donde puedes especificar ese tipo de cosas.
- Si, pero igual sería bueno tenerlo para que yo pueda elegir algunos módulos, alguna cosa así, porque en general en el proceso de la DevOps, en algunos casos no necesito tener… o mejor dicho, se sube todo el repositorio completo pero cuando se refiere a lo que yo necesito, tengo por ahí el caso de un banco (fuera de GeneXus) que tiene varios módulos y ellos hacen Deploy dos veces al día, y ellos dicen “ok, voy a hacer de 50 programas, 80, 200 programas, los que sean, entonces lo pueden partir… a lo mejor sería bueno hacer algo también.
- Sí, seguro, pero lo que quería decir es que no es en este nivel que se hace esa partición , hay dos

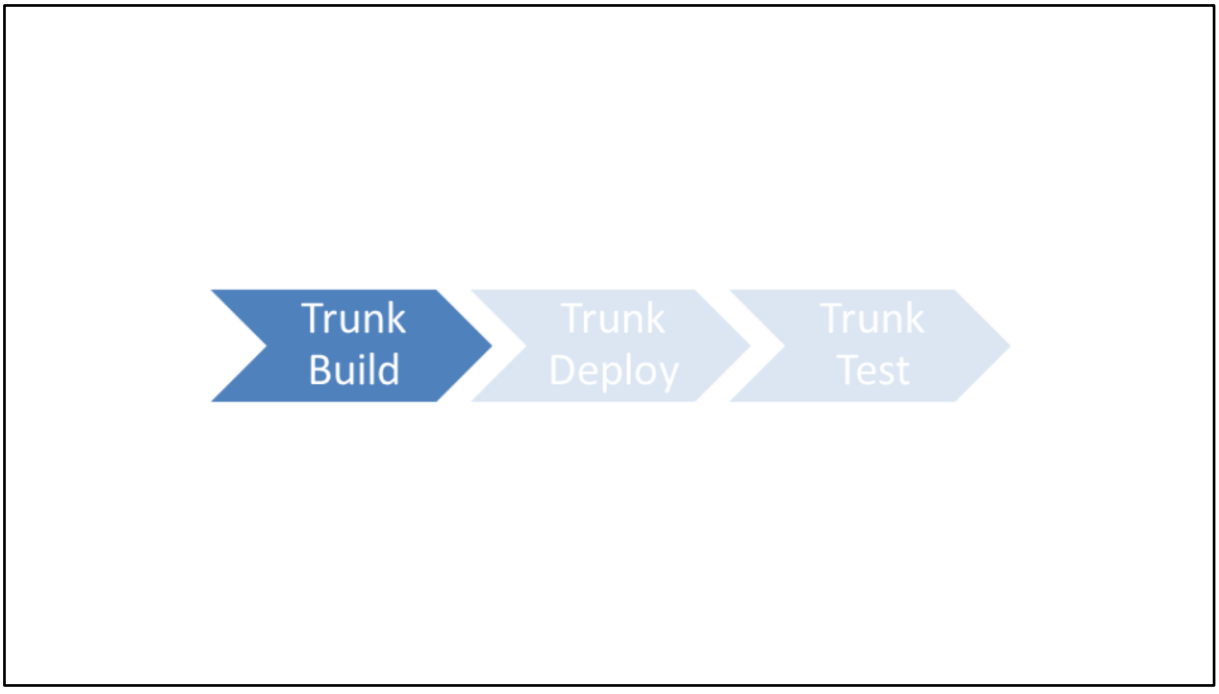
lugares en donde puedes hacer la partición, la puedes hacer a nivel de la arquitectura del código y entonces separar en diferentes KBs, o la otra opción, es a nivel de Deploy o a nivel de Build, tu pudes decir "el proceso de Jenkins está enganchado a cierta versión, cierto environment de la KB, pero el Build es solamente de éste módulo o de los objetos que sean necesario o de las cosas que sean necesarias para estos objetos… dije dos pero en realidad son 3, tu puedes particionar a nivel de código, puedes particionar al momento de Build y pudes particionar al momento del Deploy, cuando vas a hacer el Deploy también puedes decir "bueno aunque se haya armado toda la KB, en este proceso lo que hago es Deploy solamente de estos componentes a tal lugar y estos otros a tal otro, o nada" ¿de acuerdo?

Todos los ejemplos están en las opciones por defecto en las más básicas, después si tienen dudas podemos ir viendo los más avanzados…

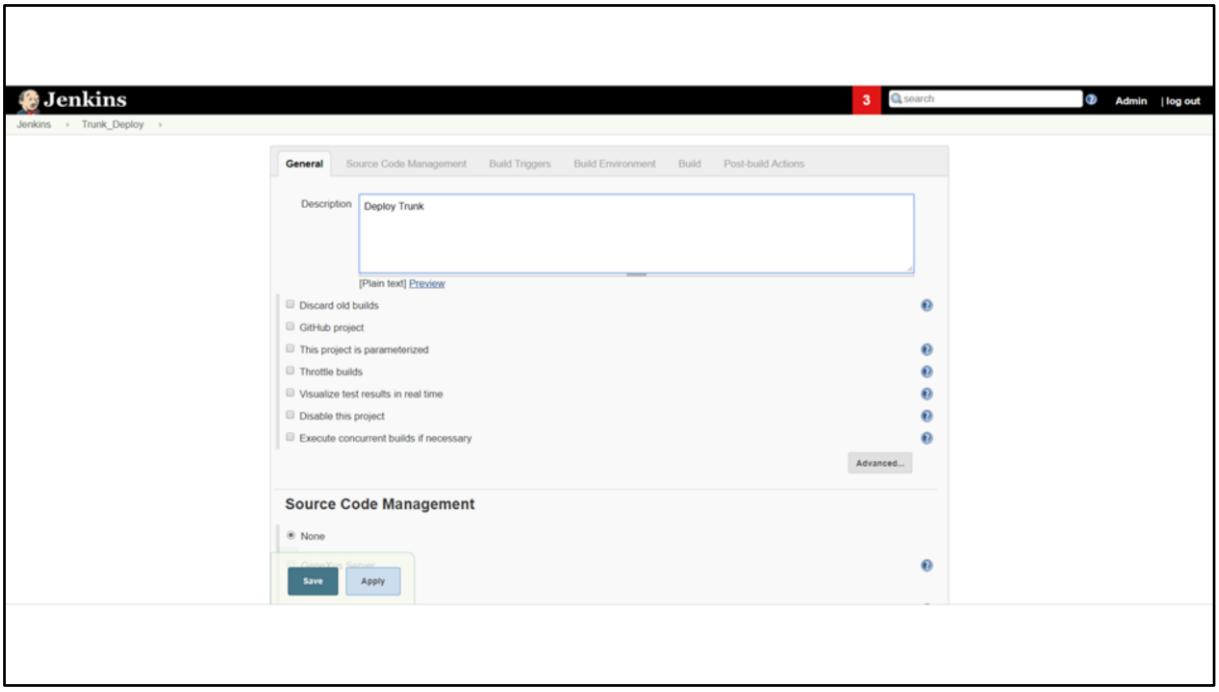
Nuevamente dejamos la versión por defecto y al ambiente le vamos a poner el ambiente que estemos utilizando.



Si vemos las opciones adicionales, podemos ver que al momento de Build podemos elegir si queremos la opción de SR Build All, en vez de Build All que es la opción por defecto.
Ahí le damos a la opción guardar y con eso concluímos la creación de nuestro primer proyecto en Jenkins.



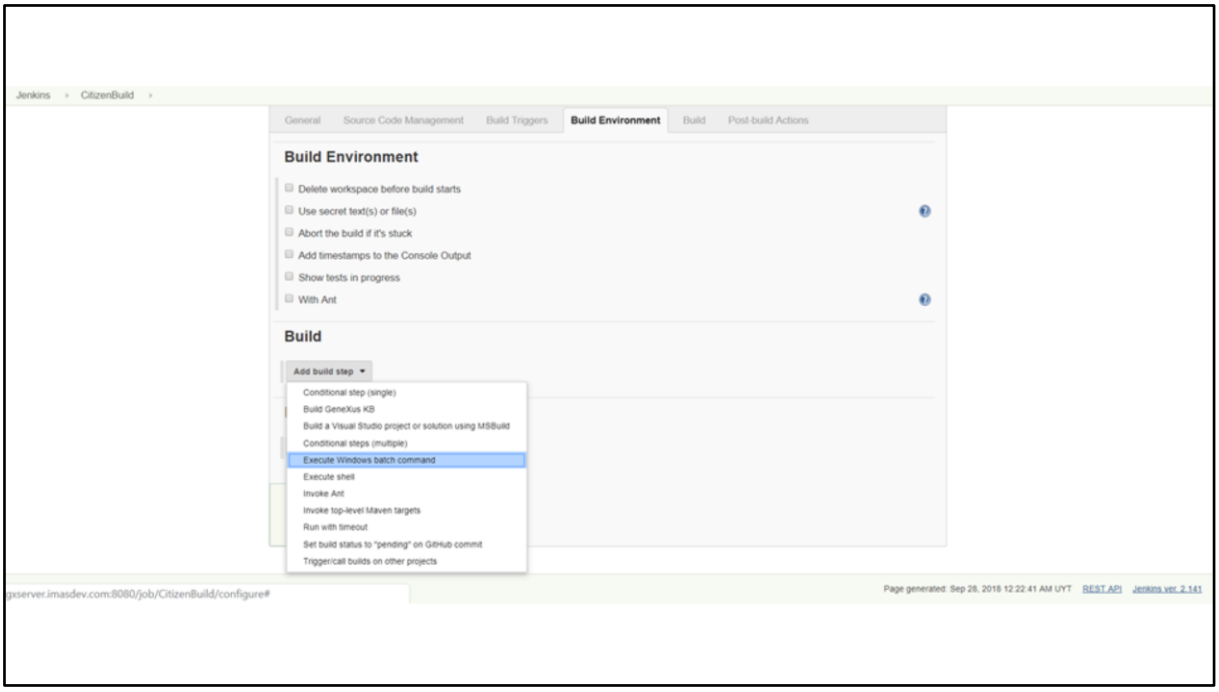
Éste proyecto lo que hace entonces, es hacer el Update desde el Server y luego ejecutar el Build All de ésta KB.



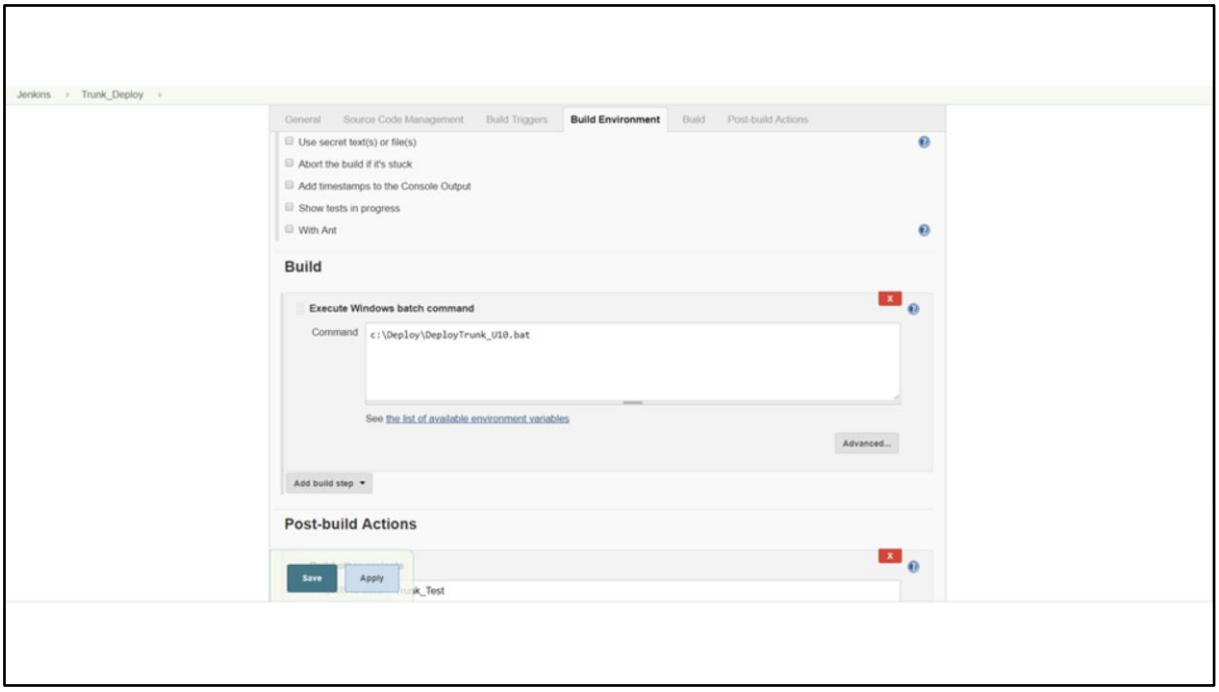
Vamos a ver ahora cómo configuramos la segunda tarea o el segundo proyecto en Jenkins que es el que haría el Deploy, es decir, el que movería los binarios generados en la KB de Build hacia el ambiente de Test…

(este también es un ejemplo súper básico, definitivamente acá lo único que voy a estar haciendo es llamando un Script, pero definitivamente se pueden hacer cosas más elevadas, esto es simplemente para hacer el ejemplo más simple)

para ello creamos un nuevo proyecto que lo vamos a llamar “Deploy Trunk”



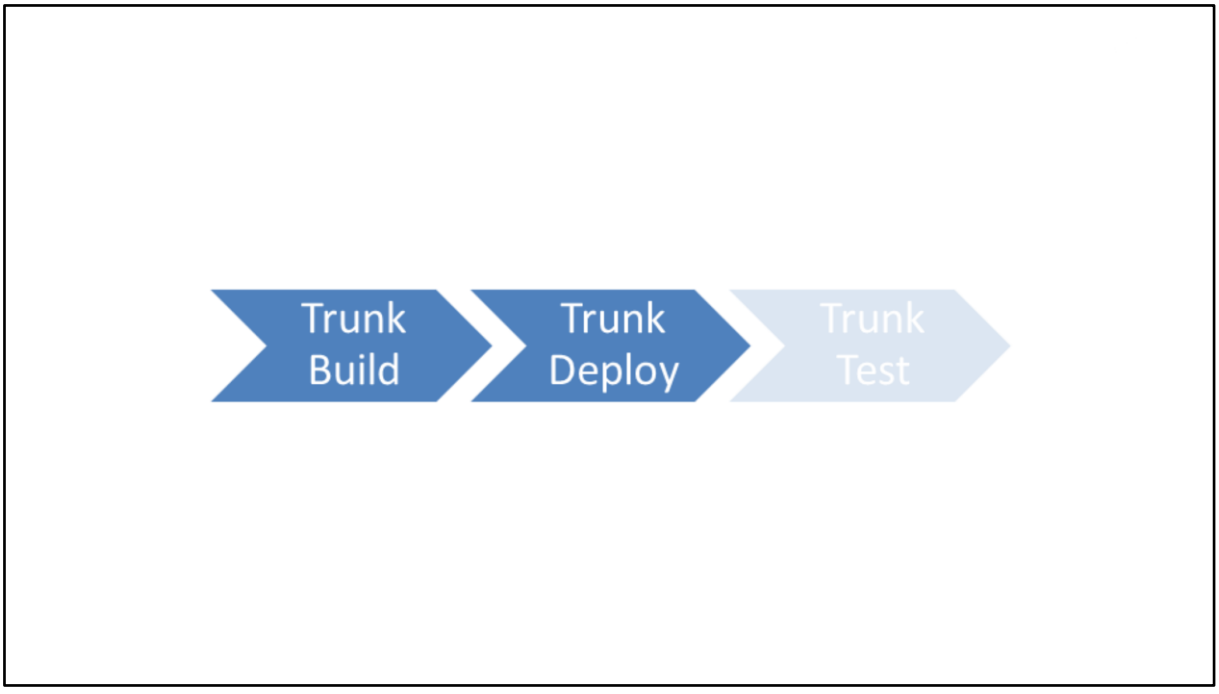
Entre las opciones de Build tenemos la opción de ejecutar un comando batch en Windows…



y vamos a configurar entonces que en el caso de Build queremos ejecutar un batch a un script, éste script lo que estaría haciendo es automatizando las tareas necesarias para publicar los binarios generados en el ambiente de Test.

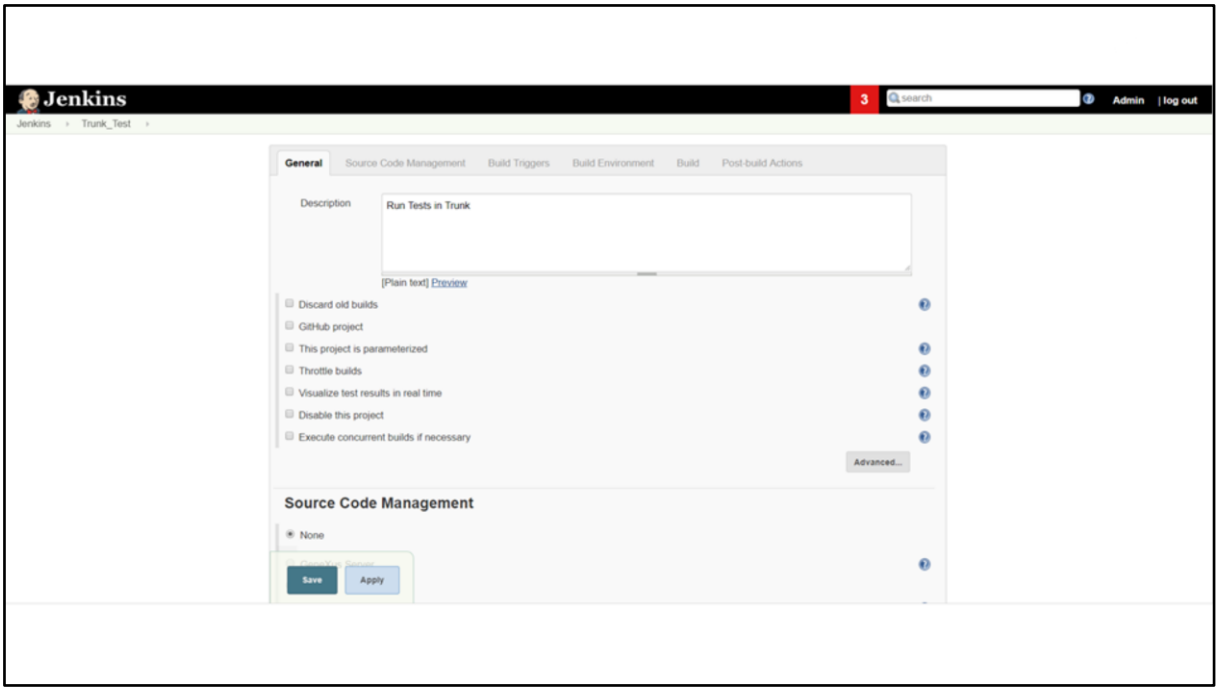
Luego elegimos la opción grabar y ya queda completada nuestra segunda tarea en el pipeline en el desarrollo del Trunk.

(Digo los binarios, pero es todo… todos los objetos necesarios para hacer el Deploy)

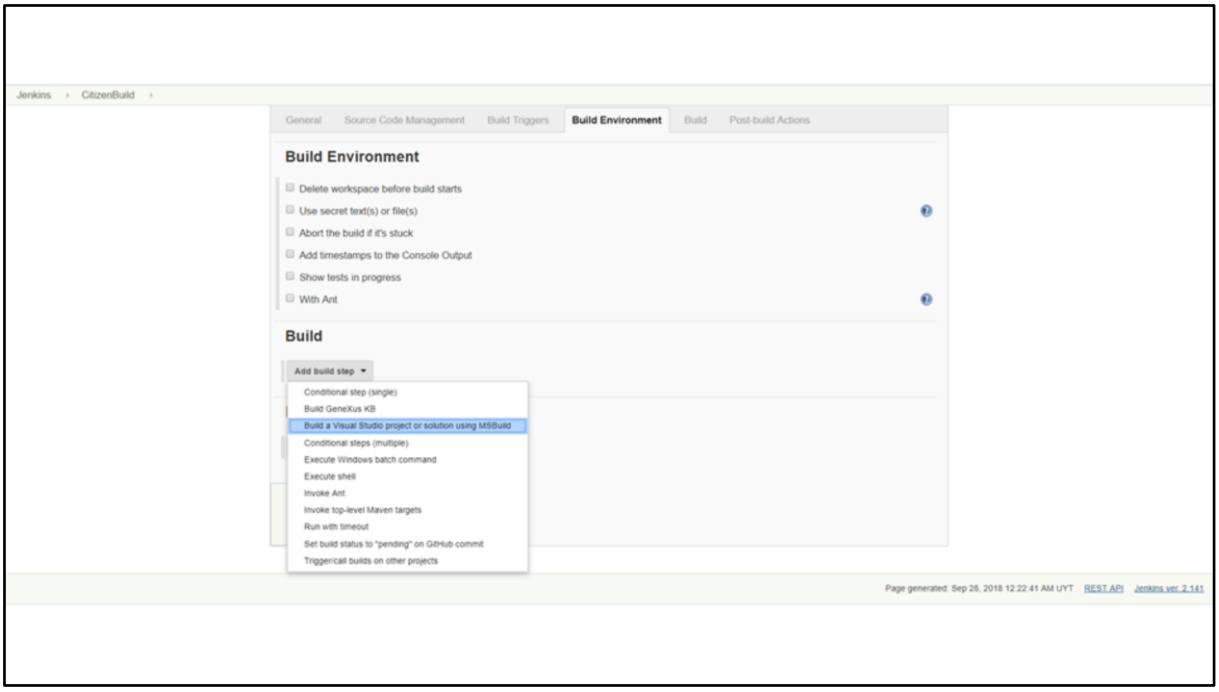


Esta tarea es la que publican los binarios generados en la KB del Build hacia el ambiente de test.

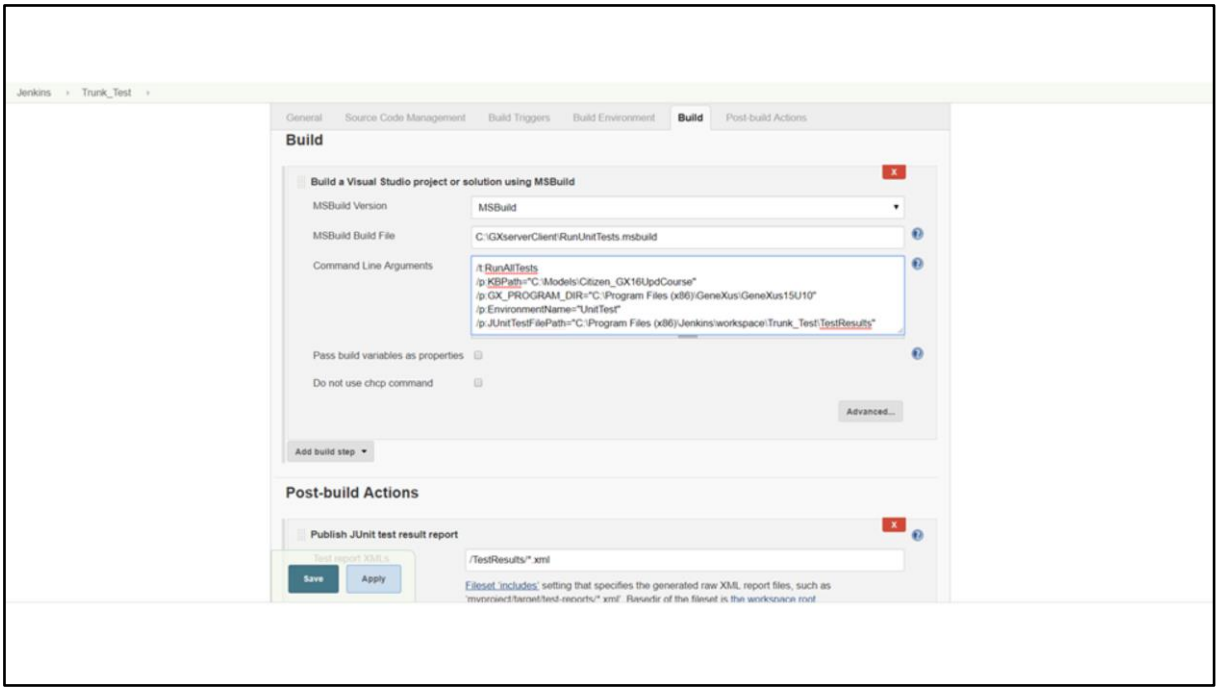
En caso de haber definido pruebas unitarias, podemos definir nuestra tercer tarea en el pipeline, que es la que ejecuta estas pruebas unitarias.



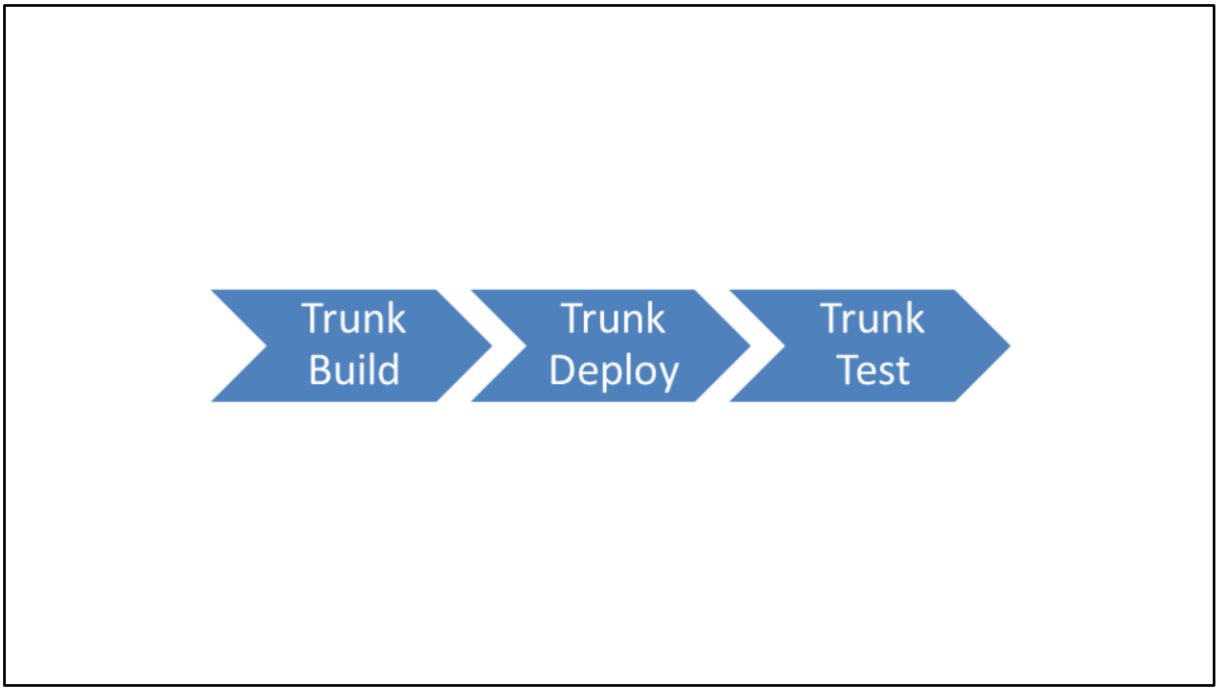
Para ello creamos un tercer proyecto en Jenkins que lo vamos a llamar “TrunkTesk” y la descripción es “Run tests in trunk”



Y ahora en la opción Build vamos a seleccionar la opción “Build a Visual Studio Project or solution using MSBuild”, porque para poder ejecutar las pruebas unitarias necesitamos hacerlo mediante un MSBuild.



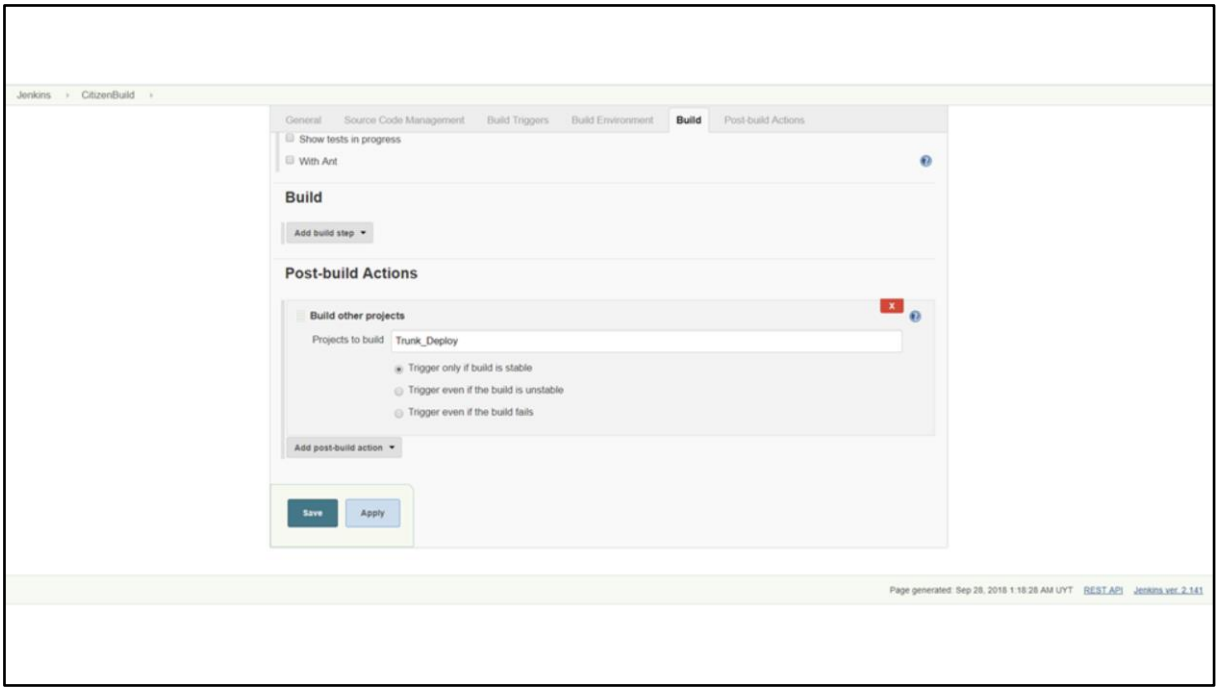
Luego configuramos el MSBuild que nos permite correr, ejecutar todas las pruebas (no vamos a entrar muy en detalle en cómo se configura, sino que pueden ver toda la documentación en el Wiki.



Pasa “volando”, simplemente para que tengan una idea de los tipos de cosas que se pueden hacer, la idea no es que salgan de acá siguiendo este video, está documentado en el Wiki, sí siguiendo los pasos del Wiki lo pueden hacer.

Definimos así, los tres proyectos en Jenkins que representan cada una de las tareas que tenemos que ejecutar en nuestro proceso de integración continuo, donde el último paso es la ejecución de las pruebas unitarias que nos permiten validar el armado del Build.

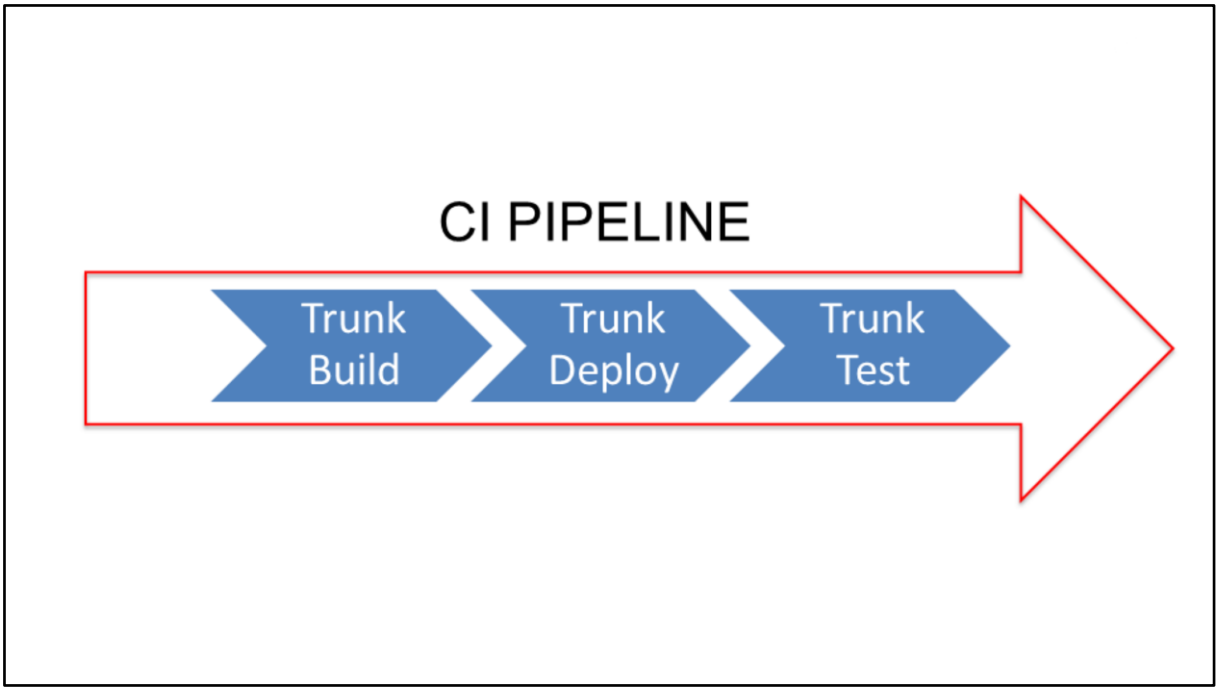
Vamos a ver ahora cómo hacemos en Jenkins para encadenar estas tareas de manera que se ejecuten una detrás de la otra.



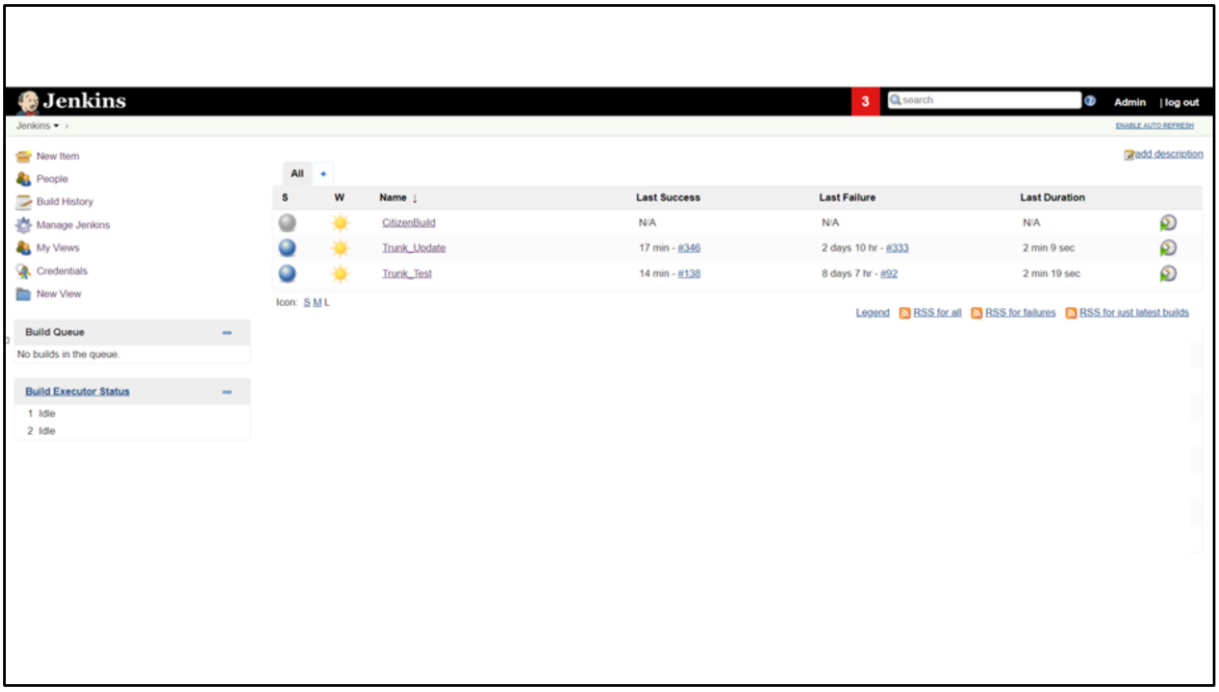
…(otra nota más ahí, es que el Deploy y el Test no tienen porqué ser una detrás de la otra, perfectamente pueden ser en paralelo, podría ser que yo digo “bueno hago los Test y no hago el Deploy si me dieron mal”, en realidad ahí también son decisiones que uno podría decir “si los Test fallan no hago el Deploy, pero en realidad mi ambiente de test ya no refleja más la KB, porque la KB es algo distinto, en definitiva ahí son decisiones de cómo trabajen, están puestos uno atrás del otro pero perfectamente puede ser en paralelo y además podemos cambiar el orden según nuestra metodología)

Para ello, vamos a modificar esta tarea de CitizenBuild.
Vamos al ultimo sector “Post-build action” y allí vamos a agregar la opción “Build other Projects”
Aquí ingresamos el nombre del proyecto que debe seguir a este (“Trunk_Deploy) en nuestro caso, luego de CitizenBuild viene Trunk Deploy que es la tarea que actualiza el ambiente de test, podemos seleccionar cuáles son las condiciones para disparar este post-build action, que podrían ser sólo dispararlas si el build es estable (build is stable), es decir, si no hubo problemas al hacer el Update del Server o el Build All, entonces ejecutamos la tarea que realiza el Deploy.

Podríamos elegir también ejecutar el paso siguiente por más que el Build sea inestable o que falle.

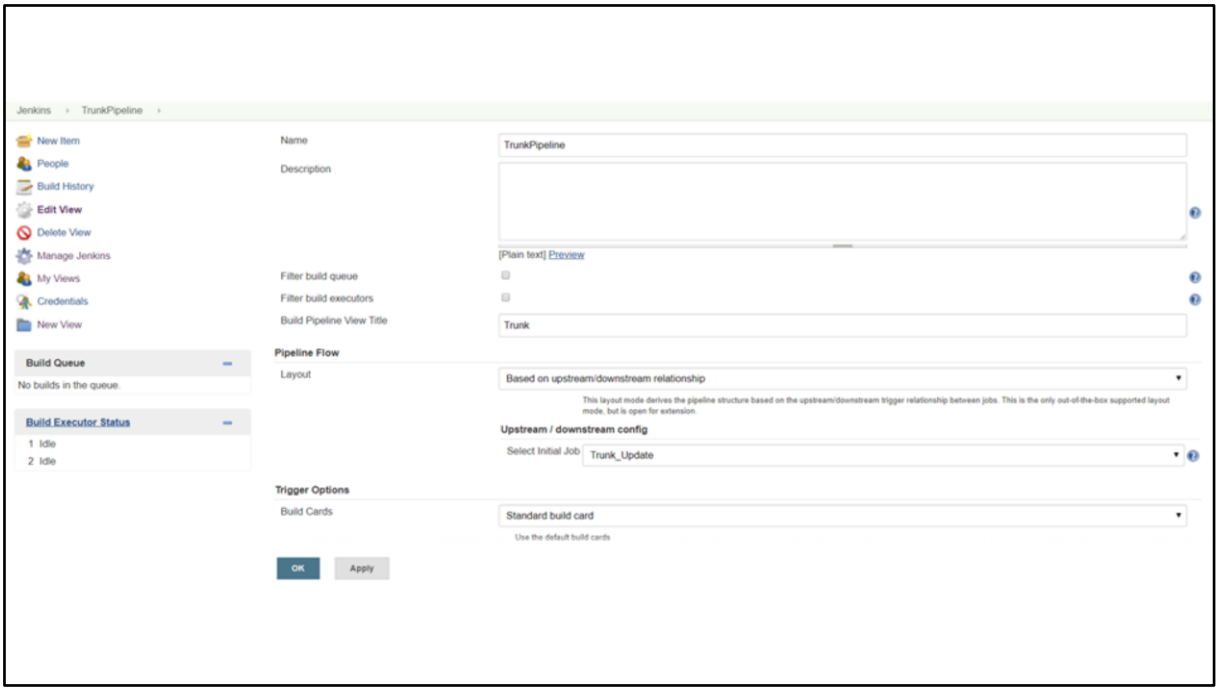


Ahora que hemos definido nuestros tres proyectos, nos queda ver cómo hacemos para visualizar estos tres proyectos en un Pipeline.

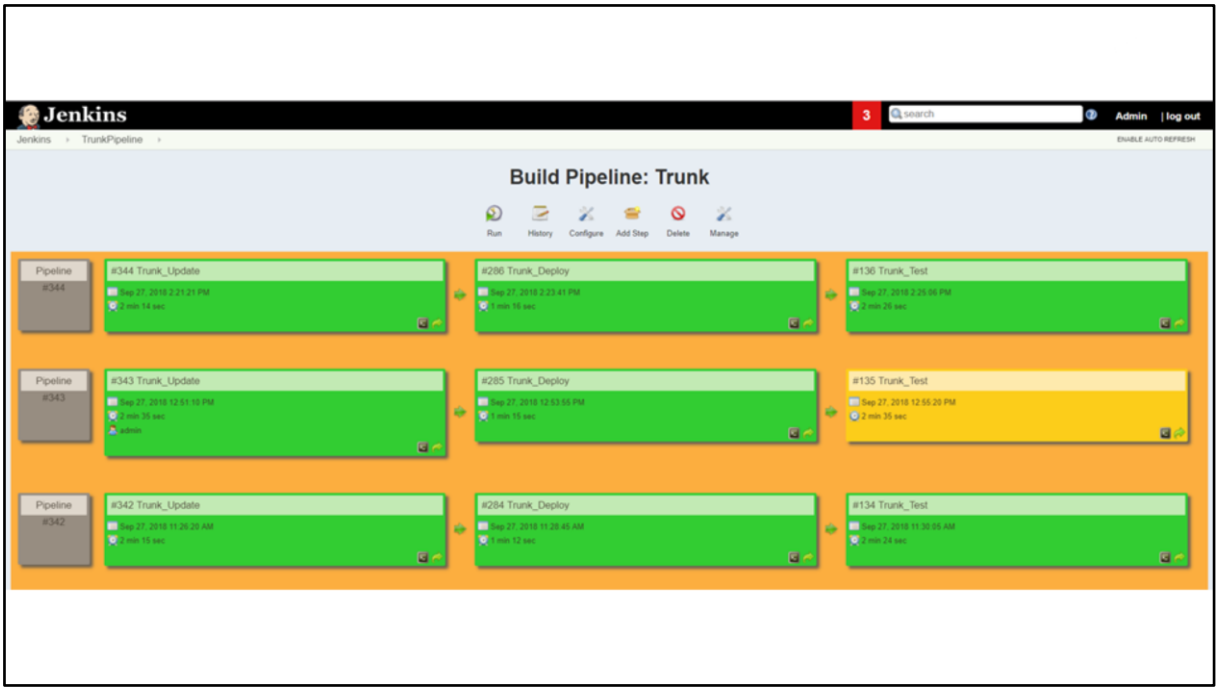


Para eso vamos a ir a Jenkins y en la página principal tenemos la opción de crear nuevas vistas.

Al crear una nueva vista, vamos a ponerle el nombre Trunk Pipeline, vamos a seleccionar que es un Build Pipeline View.



Le vamos a dar el nombre y vamos a seleccionar cuál es la primer tarea para ejecutar este pipeline, que en nuestro caso es Citizen Build



Esto que vemos ahora es la vista de Pipeline de un proyecto existente que tiene la misma secuencia de tareas que acabamos de configurar para nuestro proyecto, con la salvedad que el primer paso se llama Trunk Update y no Citizen Build.

Cada una de estas líneas corresponde a una ejecución del Pipeline, es decir, cada 5 minutos nuestro Jenkins está monitoreando el Server en busca de cambios, cuando encuentra, ejecuta una corrida de éste Pipeline, cada una de nuestras tres tareas encadenadas…

(¿Está claro? Ahí lo que estamos viendo es la historia de los últimos “x” cantidad de Builds, la que está arriba es la última que se acaba de ejecutar, cada corrida es una línea, hacia abajo tenemos las anteriores, en el Build #345 ese tiene el último commit, el #344 es el del commit anterior o de hace 5 minutos)

…Para cada una de ellas podemos ver los resultados o la salida de la ejecución de la tarea.
En el caso de la tarea del Build, lo que tenemos es la ejecución del Output de GeneXus.

En el caso de la tarea de Deploy, lo que tenemos es el resultado del Script que está haciendo la publicación.

Y en el caso de los Test, vamos a tener la salida de los assert que quizás es un poco difícil de ver, pero podemos ver en los Status los Test Result y ver que los Test estén todos pasando.
Incluso podemos ir a ver la historia de los Test, en donde nos va mostrando en las distintas ejecuciones cuándo hubo errores y cuándo no.

Participantes:

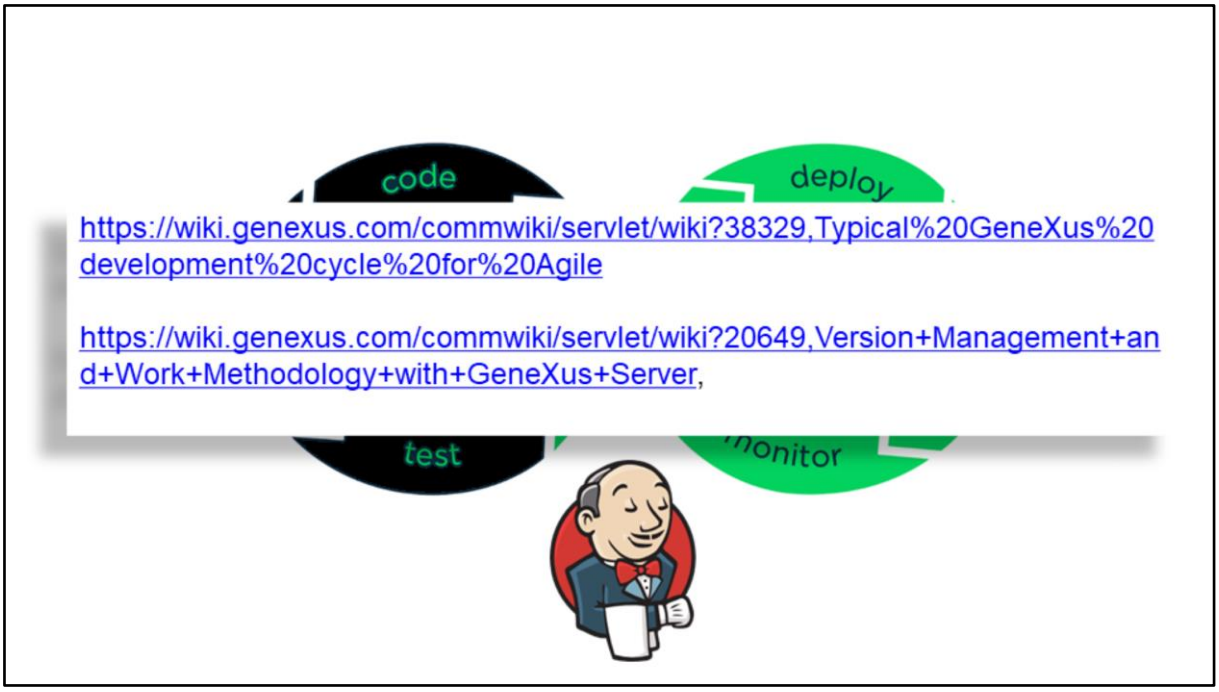
- Mi pregunta va por el lado de que voy llevando el control de todo el procedimiento ¿hay alguna conexión en algún momento de Jenkins con el versionado que tiene GeneXus, a medida que yo voy haciendo todos estos procesos, ya sea con mi versión 1 de productivo, la marco como versión? ¿hay alguna conexión o tendría que llevarlo todo acá en Jenkins?
- En realidad técnicamente no hay una conexión con el versionado de GeneXus, pero técnicamente cada corrida de esto genera un Build de tu aplicación, o sea, que ahí vos podes versionar eso…
- Sí, no hay nada directo, pero lo que estamos mostrando aquí son el proceso canónico, lo más standard, pero por debajo lo que estamos ejecutando son tareas MSBuild y existen tareas MSBuild

específicas, por ejemplo que tienen que ver con versionado, uno puede hacer commit, puede hacer congelado de versiones y ese tipo de cosas.
Entonces, así como se hace el correr… Lali decía " se pueden correr los Test" y en base a eso tomar la decisión de si hacer un Deploy a otro lugar, también se podría tomar la decisión de ejecutar otro paso que lo que haga sea congelar una versión o hacer un commit, ese tipo de cosas…

- Claro, a eso iba yo, porque de alguna manera a lo mejor ustedes podrían implementar un plug-in que se pudiera hacer un congelado en este momento, y yo después voy por GeneXus veo mi versionado y me encuentro con ya actualizado…
- Pero eso ya lo podes hacer con un MS Build.
- Sí, eso se puede hacer ya llamando al MSBuild, tu puedes especificar exactamente ese tipo de cosas, esto es siempre un balance, porque por el lado del MSBuild es la línea de más bajo nivel, entonces ahí tienes la libertad total de hacer lo que quieras.
  Por el lado del Plug-in, lo que tratamos es de levantar el nivel de abstracción para que por lo menos las cosas más comunes te sean mucho más fácil, algunas de esas podemos incorporarlas también en el plug-in con algunas opciones y entonces así como hay un Build Step, que es armar una KB y simplemente dices eso, también podríamos tener un Build Step que fuera congelar una versión, hacer un "import" o ese tipo de cosas **,** eso se puede ir haciendo, de hecho también el propio plug-in es código Open source, es decir que también contamos con que algunos también van a querer meter mano, colaborar y agregar cosas como esas también.
- Ok, gracias!
- Si, este Pipeline es el Pipeline de desarrollo, después en realidad este que están viendo acá es el Pipeline con el que trabajamos en mi empresa, en donde los Test se están ejecutando en la misma KB que hace el Build, en un ambiente separado que está conectado a una base de datos local y demás, en realidad esos pasos corren en paralelo.
  El Deploy hace el ambiente de Test que es un servidor aparte y la ejecución de los Tests unitarios.
  Pero en nuestro Pipeline de release ahí sí tenemos, hacia el final cuando se va a producción por ejemplo, se hace un freezado de una versión y se actualiza a una KB que tenemos siempre con el código vivo, o sea que sí, la complejidad del Pipeline va a depender también mucho de tu negocio y de cómo sea el proceso, pero tenés mucha flexibilidad en qué cosas implementar.
- ¿Cómo queda la reorg en ese proceso?
- En realidad, acá mismo la reorg está corriendo desde la KB… la misma respuesta que al tema del versionado, por ejemplo, en nuestro Pipiline de release, lo que tenemos es que hay un paso que va y captura la reorganization TXT y lo pone en un folder para que después nosotros sepamos que a partir de ahí tenemos que ejecutar las reorgs.
  Ahí también va un poco en las herramientas que manejen y demás, Jenkins lo único que hace es orquestar cosas.
  Nosotros en realidad capturamos los reorganization txt, pero se podría agarrar el programa de la reorg o se podrían hacer cosas distintas.

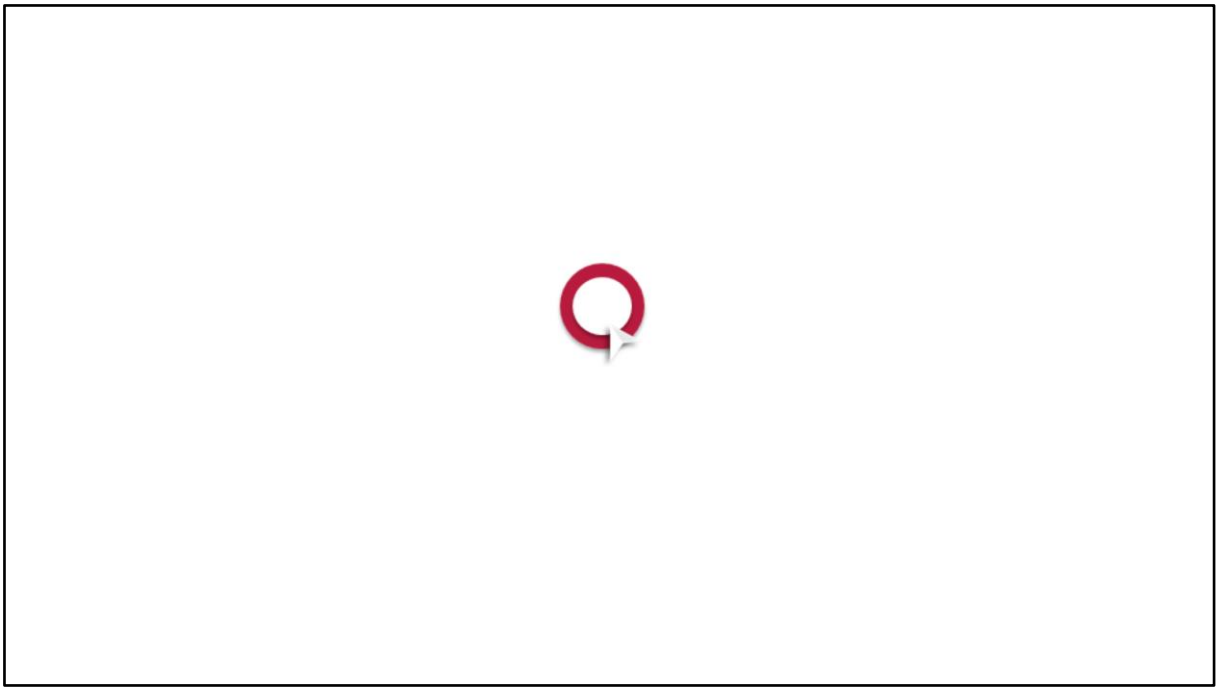
Si queremos realizar una nueva corrida en nuestro Pipeline, le damos al botón "run" y eso sin esperar los 5 minutos a que el Server lo haga solo, en ese caso se ejecuta una nueva corrida.
Aquí podemos ver cuando Jenkins se está ejecutando la tarea está en amarillo y mientras las tareas todavía no se han ejecutado quedan en celeste.

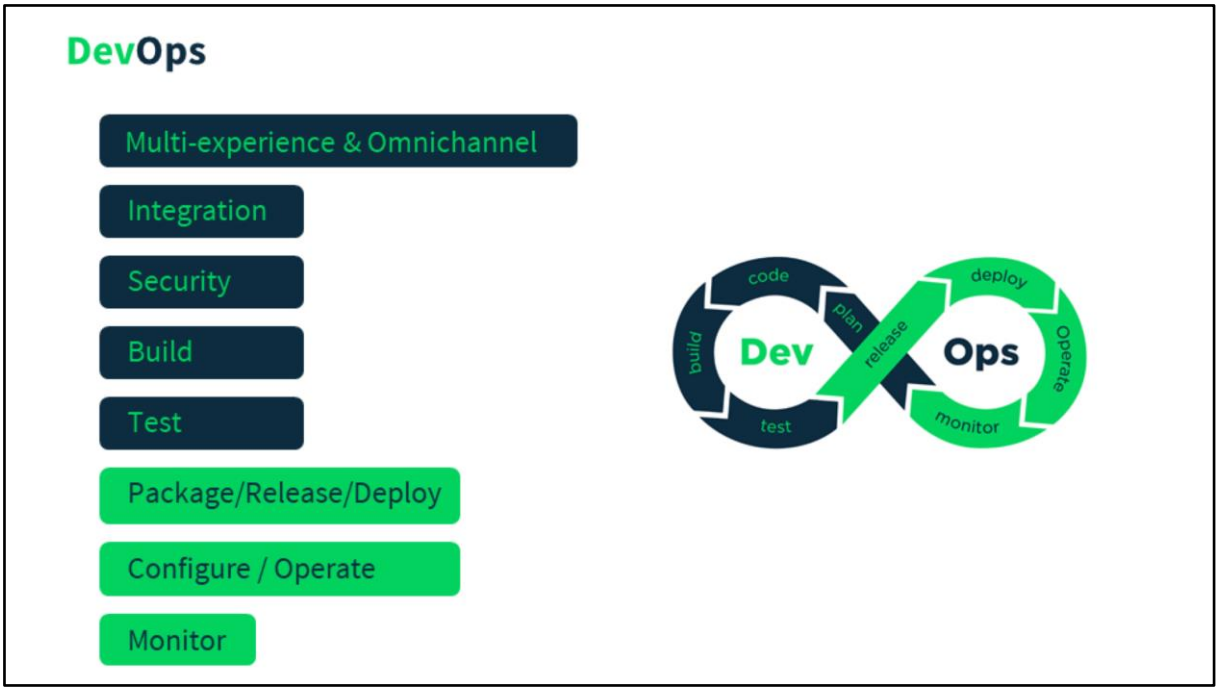
Otra cosa que podemos ver gracias a la integración entre GeneXus y el Jenkins, es que cuando vemos los detalles del proyecto del update, podemos ver cuáles son los commit que están asociados y podemos ver la información de quién hizo el commit y la descripción que se le dio al mismo.
Incluso si vamos a ver los detalles del Commit podemos ver cuales son los objetos que fueron modificados y que en definitiva estuvieron incluidos en este proceso de Build.



Hasta ahora hemos visto cómo GeneXus, GeneXus Server y Jenkins nos permiten ejecutar las primeras instancias de este ciclo de DevOps, que son las que corresponden a la integración continua.
La fase de release implica una decisión con respecto a la estrategia de manejo de versiones y metodologías de trabajo, hay documentación de esto en nuestro Wiki.
Posiblemente tengamos otro Pipeline de integración continua implementado en Jenkins, que automatizará los diferentes pasos en manejo de versiones y también el paso Deploy a producción.

Es importante notar que no todos los pasos en Jenkins tienen porqué ejecutar de forma automática uno atrás del otro.
Quizás nuestros procesos no estén tan aceitados para ir a producción sin un test en diferentes etapas, al definir cómo se encadenan las tareas, podemos indicar si la misma se dispara en forma automática o si se dispara en forma manual, por ejemplo podríamos ejecutar el Pipeline de release para el armado y publicación de un ambiente de QA y quizás dejar el Pipeline en pausa hasta que el equipo de Test nos de el ok para seguir avanzando.





A medida que repasamos todo el ciclo de DevOps, hemos visto los cambios más importantes entre la versión 15 liberada y la versión 16.

¿Preguntas?

- ¿Cuáles son el tipo de fallas que puedo encontrar una vez que se ejecutan estos procesos? Por ejemplo, hace un rato mostraron una lámina en la que aparecía “aquí tengo procedimiento 1, procedimiento 2” y que salieron bien, ¿cuál es el tipo de fallas que podrían presentar desde el lado de GeneXus?
- Ahí cada una de las tareas tiene que resolver si terminó exitosa o no, en el caso del Build de GeneXus, en realidad falla cuando (si tu haces un Build como un usuario en GeneXus) si falla la reorg, si algo no especifica, si algo no compila… no te captura nada funcional, para eso tenés una etapa en que se corren pruebas, pero el armado de la KB y las fallas posibles son esas.
  No puedo hacer un update del server porque hay un conflicto de algo, no puede hacer la reorg, no especifica o no compila…
- Ok, lo último… hay herramientas que pescan aparte lo que es la generación de las aplicaciones, como en este caso Jenkins, ¿pero hay algunas herramientas que son complementarias para lo que es el seguimiento de tickets, para ver qué es lo que pasó? Si ahí dicen, mira tengo 20 tickets, yo se los puedo asignar a diferentes personas ¿cuándo tiene pensado incluir algo de eso y con qué producto tal vez?
- Lo tenemos en el radar, el otro día en las charlas sobre plataforma decía que justamente la línea en la que estamos yendo es a que GeneXus Server sea “server” en un sentido más amplio, inicialmente el foco de GeneXus Server es la parte de SCM, el tema del versionado, trazabilidad, trabajo en equipo, etc.
  Pero justamente vamos con la línea de incorporarle más cosas.
  Una de las cosa que estamos haciendo, es esto de Jenkins, integrar en un sentido, es decir que Jenkins pueda conocer lo que es GeneXus y lo que es GeneXus Server, eso es el Plug-in en Jenkins que permite que entonces uno le pueda decir “esto es una KB GeneXus, haceme un build” o “esto está conectado con un GeneXus Server, haceme update” y que lo vea como una cosa nativa, así como lo habíamos hecho en su momento con Cruise control. La idea es también integrarlo en el otro sentido, es decir que desde GeneXus Server se pueda conectar con procesos como estos de Jenkins y entonces si bien siempre vas a tener la chance de ir a Jenkins y configurarlo tu a mano, hacerlo por xml, etc. La idea es ofrecerlo también desde el lado de GeneXus Server, que en la consola de GeneXus Server o

incluso quizás en el IDE de GeneXus puedas decir “quiero definir un proceso de armado sobre esta KB, con tales condiciones, etc.”
Y de la misma manera también, así como hablamos de integración con Jenkins, integrarnos con otras herramientas como puede ser para Code Review, hemos estado trabajando con Fabricator también, haciendo algunas pruebas y para Issue Tracking hay otras herramientas también. Hay múltiples herramientas y todas tienen mecanismos de extensibilidad y plug-in que permiten conectar unas cosas con otras, es decir Fabricator con Jenkins, Jenkins con Fabricator, con GeneXus, etc. Y la línea es ir así, en ese sentido.
De vuelta, tenemos siempre dos canales, uno es que existan los componentes aunque sean de bajo nivel pero que te permitan toda la potencia, no frenar a nadie, si la potencia está… claro, hay que meterse un poquito a más bajo nivel y en la otra línea trata de que no necesiten toda esa complejidad y que por lo menos las cosas más usadas nosotros las podamos dar con más alto nivel y más fácil.

- Disculpa, no se si me perdí en algún minuto… en general se hablaba de GX Server, ¿y si no tengo GX Server?
- El GX Server va a ser el orquestador de todo ese tipo de cosas, lo que quizás puede ser que tu puedas tener licenciado o contratado algunos componentes y otros no, no es que tengas que tener GX Server para todo, para nosotros técnicamente GX Server es el que orquesta todo, lo que quizás puedas tener el componente que te da continuous integration o que te hace Build automático y no tienes el que te maneja Code review o Issue Tracking, pero estamos hablando técnicamente, comercialmente cómo se va a fraccionar eso y comercializar no lo tengo claro todavía.
  Pero sí, todas estas cosas no las definimos nosotros, el que tengas que tener GeneXus Server en el sentido de SCM del versionado del control de código sí, eso es la práctica en la industria, es decir, hay que usar un repositorio de código, hay que usar versionado, hay que hacer commit, hay que hacer versiones, nosotros lo que hacemos es dar la herramienta para eso, pero que lo necesitas sí, claramente lo necesitas.


GeneXus™
The power of doing