Deploying and Executing SSIS Packages in a CICD Pipeline

Chris B. Behrens
SOFTWARE ARCHITECT
@chrisbbehrens

Understanding Synchrony

Synchrony

The state of operating or developing according to the same time scale as something else.

Phase One Synchrony

Things don't get out of sync very often

So you just synchronize them manually

Phase Two Synchrony

Now, de-synchrony happens every night

So you pull the code from version control...

Upgrade the target schema...

And execute the package

You could do this manually, but why?

Maturity Level Three

This level requires changes across the board in your organization

We're keeping two or more things in concert with each other

The executing version...

The version in version control...

And the database with all of this

Immutable Server

A server that once deployed, is never modified, merely replaced with a new updated instance.

The Challenge of Transactional Databases

Reasonable to achieve with web or application servers

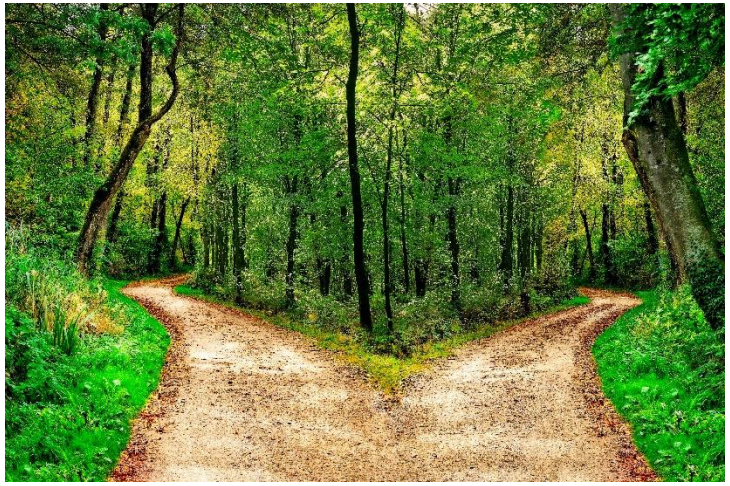
Much, much more difficult with databases

For more on this, see my previous course

https://app.pluralsight.com/library/courses/microsoft-azure-web-applications-services-deploying

Variables in SSIS

Deterministic File Paths



Employee file comes from path A

Activity file comes from path B

These two paths are probably the same except for the file name

When they're properties, they're entirely independent of each other

It's better if they're both functions of a variable and filename

Demo

Briefly review the new activity completion logic

Modify our two file path connections

To point at a single variable

Xml configuration

Command line argument

Making This Work in a Build Pipeline

Our Jenkins Architecture

Jenkins is installed natively on my Windows laptop

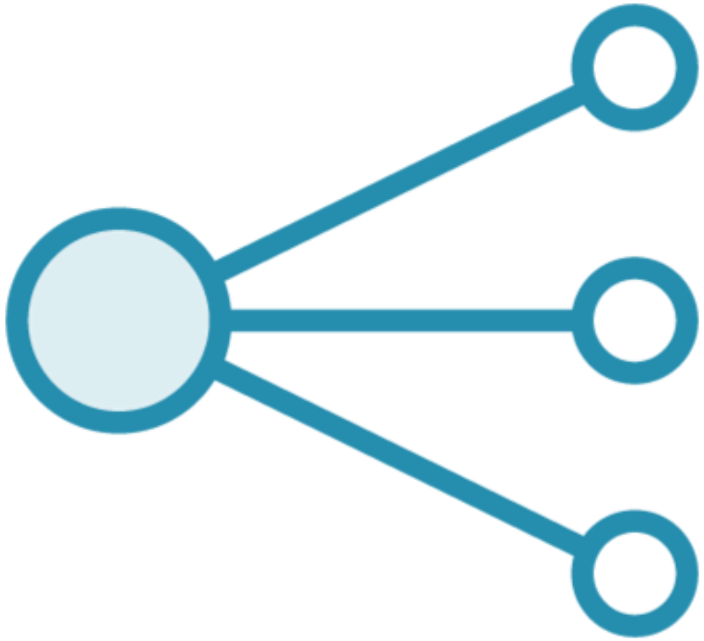
Running in a single master/agent config

In real life, you'll spin off child agents to spread the load around

The architecture is Windows, Windows, Windows

In real life, this should probably be Linux

Our ETL Architecture



DTEXEC and the other data tools must be installed on the build server (or the agent in an agent config)

Remember to make sure the versions line up with each other

Where is the Sql Server running? More on this later

Demo

Create a very simplified and streamlined build

- To demonstrate the minimum concept of executing a package in a pipeline

Create a Jenkins pipeline

- Pulls our package from Github
- Executes it

Look at the data in the target instance

A Security Problem with This Build

There are two security issues here

The first is the credential set used to connect to the target db

Which you need to test the package

Package Password (non)Protection

Don’t panic – it’s protected by a password

Exercise for the learner – write a brute force attack console app against the package xml

The Second Problem

The password in the Jenkinsfile is bad, too

Sensitive stuff doesn't belong in version control – more on this in the last section

But if we fix the first problem, the sensitive data in the package...

We don't have to protect the package at all

Encrypt with User Key

Your package has a property, ProtectionLevel

EncryptSensitiveWithUserKey

Encrypted with a machine-specific key

- So we can't use this in our build process

The companion value EncryptALLWithUserKey has the same problem

Encrypt with Password

Encrypt your content with a package password

EncryptSensitiveWithPassword, or EncryptAllWithPassword

Without a server, there can be

- No lockout
- No expiration

Don’t use any of these previous methods

DontSaveSensitive

Sensitive data is simply not stored in the package

Because it's never in the file, it's never in version control

These values have to be supplied by the execution context

But this also means that you have to re-enter the password(s) whenever you re-open the project

Demo

Set our package protection level to "don't save sensitive"

Modify our build to inject the password

- Using a helpful tool
- Execute our build
- And review the results

Package CICD Wrap-up

We haven't really fixed the problem

Because the password is in the Jenkinsfile, which comes from version control

Use environment variables to inject the password into the build

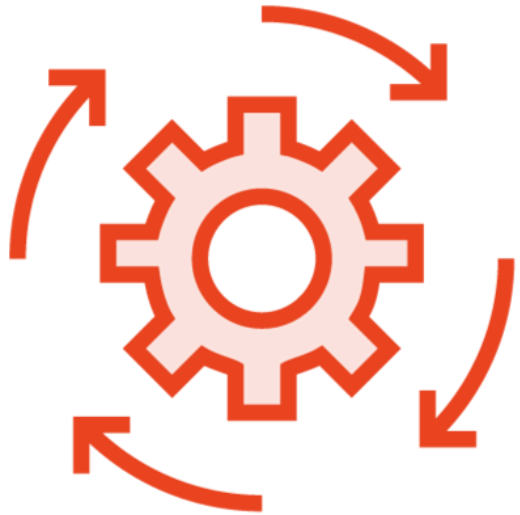
In DevOps, use secure parameters or even a Key Vault

Get Sensitive Data out of Version Control

Once everything is out of version control...

Attackers can get your code, and you don't care

This is going to have to happen no matter what you do



So that you can have generic builds

Build Triggers

The trigger should be automated in one way or another

It can be scheduled

If so, you should verify the existence of the incoming files

This is usually easier to do in the build than in the package

Summary

Variables

- An expansion of our work in setting properties
- Expressions, how we are able to drive property values
- Variables we create
- Variables that are created for us, like system variables

Automating our package with a build server

- We got a simple build running
- Using DTEXEC
- Relocate our sensitive information
- Get it out of our package