4 Simplistic Ways To Improve Your Software Deployments.

Craig Godden-Payne
Jun 9 · 5 min read ★

This article will refer to a hypothetical deployment process of:

1. Build software artefacts.
2. Zip artefacts.
3. Upload artefacts to the server using SFTP.
4. Send email notifying users of the restart.
5. Stop webserver.
6. Tail server logs to check for the stop.
7. Backup files on the webserver.
8. Move new artefact files into the webserver.
9. Start the webserver.
10. Tail the server logs to check for success.
11. Send an email notifying users of completed restarted.

It seems like a pretty straightforward deployment pipeline.

Unfortunately, we usually have to work with pipelines much more complicated, but we can interchange pretty much any deployment pipeline with the above using this guide.

Regardless of how nasty the current deployment process is, always work with small steps instead of taking on the whole challenge.

The Only Step You Need to Progress in Your Career as a Developer or Engineer.
I don't usually like to write articles about myself, as it feels a bit self-indulgent, but I thought it would be useful...
medium.com

. . .

Step 1 — Stop performing manual deployments.

Some companies *still* deploy software manually.

With manual deployments, there are not many improvements to be had, without some scripting. The first stage is to find ways to prevent manual deployments.

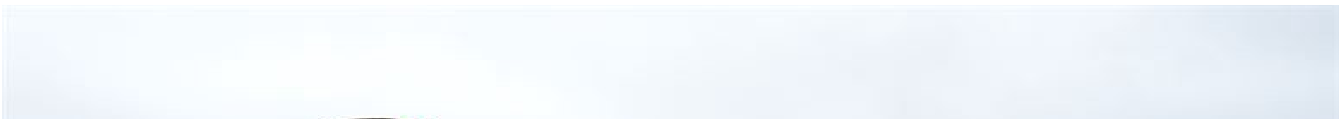
What is a manual deployment?

If it comes to release day, and the process involves manually logging onto a server and copying some files over, then it falls into this category.

There are many problems with manual deployments:

- **Spreading knowledge is a challenge;** Sometimes, one or a small number of people know how to deploy. However, what happens when the process changes or *the deployment guy* is off sick, or leaves?
- **Lack of consistency;** What is the state of the production server? It has likely got an unascertained amount of software installed, of which most are grossly out of date.

Step 2 — Script what you can and automate



It is inarguably better to script a deployment than to have a manual deployment.

By scripting, the deployment is already taking a step in the right direction. It means that eventually, it can be automated.

Why script deployments?

Using a script shows the exact steps involved, and can be understood by whoever is running the deployment.

It empowers others to be able to deploy rather than a select few.

With the example provided above, a bash or command script can automate at least some of the stages and combine them.

Creating a Simple, Low-Cost Twitter Bot, utilising Serverless Technologies.

People have a love-hate relationship with twitter bots.

medium.com

• • •

The idea is not to have a single script which arranges the entire deployment.

Consider splitting a more extensive deployment into smaller pieces and utilise stage checkpoints, so that the deployment can rerun from a particular stage.

Attempt to make the deployment idempotent so that it can run multiple times without issues.

The deployment scripts should now look something like:

1. Build

- Build software artefacts
- Zip artefacts
- Upload artefacts to the server using SFTP

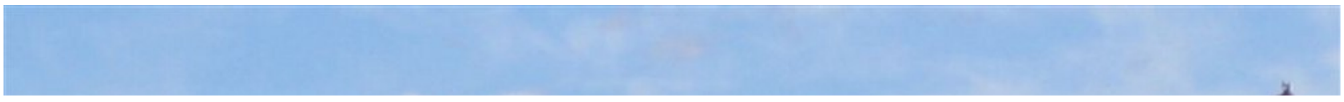
2. Backup

- Send email notifying users of the restart.
- Stop webserver.
- Tail server logs to check for stop
- Backup files on the webserver

3. Deploy

- Move new artefact files into the webserver
- Start webserver
- Tail server logs to check for success
- Send email notifying users of the completed restart.

To make the scripts more reliable, consider splitting up even further, or changing the process around to prevent the amount of downtime or improve the reliability of the Stop/Start webserver.



Step 3 — Use a proper CI/CD solution

Using a CI/CD solution to track deployments not only leaves an audit trail of what has happened.

Deployments will also benefit from other features, such as artefact versioning and backups, auto-scheduled backups, obfuscating credentials in a credentials store, etc.

There are many different CI/CD solutions out there, and each one will have its use case. Some are free, some paid for, but generally, any CI/CD solution is better than none.

A few to mention:

- Jenkins
- Team City
- GOCD
- CircleCI

Usually, CI/CD systems will be used with the scripts that have already been created, with slight modifications.

Straight away the deployment will benefit from features such as automatic build versioning, storing credentials in a credentials store, so they no longer are exposed in the deployment scripts.

Using Microservice Patterns in an increasingly Serverless world

When working on an application domain, it is beneficial to use the microservices software design pattern.

medium.com

. . .

Step 4 — Shout about your successes.

Look how excellent the deployment process now is.

The script has now deployed 100 times without a single failed deployment, and it is because of slight changes to the ways of working.

It feels powerful to be able to quickly and easily deploy, rollback, version etc., tell people about how awesome this is.

Many companies now automate most of their practices and deploy to production many times, sometimes up to 100 times a day.

By automating deployments, it will start to give other parts of the business confidence in the ability to deploy, which is empowering.

Everything starts to move faster, and since most CI/CD systems support templating, next time a new software program is deployed, the template can be reused, effectively speeding up the first iteration.

Software Development Software Engineering Web Development Design Patterns