# Develop and Deploy Faster

In order to support your business initiatives, you need to be able to react quickly to change. Your CMS integration with your website should be no different.

If *Marketing* needs a new landing page, you shouldn’t have to wait for a specialized developer or a release window.

With the right processes and infrastructure in place, you can greatly decrease the time it takes to develop a new feature and enable more frequent deployments.

Some folks even deploy to production multiple times a day!

## Find the Bottleneck

Bottlenecks can occur throughout the development cycle. The key to enabling faster development and deployment is to find and address these in a reasonable way.

A typical cycle for a new feature looks something like this:

1. Development/implementation
2. Quality Assurance and testing
3. Deployment to *UAT*
4. Deployment to *Production*

### Do you have enough Development capacity?

If you have only specialized developers that can work within the website code base, this can be an obvious issue.

The goal should be to get as many developers as possible to be able to work on your website - not just backend or full-stack developers. Allowing frontend developers to be able to seamlessly contribute to your website codebase is critical to supporting the business’s needs.

#### What if the development environment requires an OS with certain software dependencies or security requirements?

Consider utilizing *virtualization* and *containers* so that all of your developers can work on the same configuration, regardless of their client OS/configuration.

#### What if Frontend developers aren’t comfortable working in an IDE such as Visual Studio?

Where possible, avoid building solutions that depend on Visual Studio. Consider using frameworks such as .NET Core, which can be run in any IDE, on any platform.

#### What if the Frontend developers have trouble working with server/backend logic that is contained within their views?

Where possible, separate the backend and frontend into different applications. When you can’t reasonably separate them into different applications, you can also consider using JavaScript as your view engine and rendering that on the server (i.e. ReactJS.NET).

#### With more developers involved, how do you maintain the code-base and enforce standards?

Use a *Gated Check-In* policy in your source control. Developers should NOT be able to check-in their code unless it has been reviewed and approved in a *Code Review* session by another developer.

### What is your QA Process?

Your QA process should not only test any changes in your code, but it should also provide some form of regression testing on existing code across your website. When you deploy to production, you need to be confident in your release.

You can’t cut corners here. But, you can put measures in place to speed it up.

#### Are you using any Automated tests in your Code?

A combination of unit and integration tests can go a long way in speeding up your QA testing. These tests can be written by developers and immediately notify you of an issue during the build process - even before the code is released.

#### Cross-browser Automated Testing

Using tools such as *Selenium* can allow you to automate tests across all sorts of browsers, devices, and OSs.

#### How long does it take to QA your site?

Record how long this takes. Everything from automated tests to manual regression testing, it’s important to get a benchmark. Based on this, you can identify how often you can reasonably deploy to production with confidence.

#### Who deploys to UAT/Production?

Developers who are checking in code changes should not have to worry about how or when to deploy their changes. This can be yet another barrier to having them contribute to your website.

Whether you have a dedicated role for this or an automated workflow, deployments should be centrally managed. This is one area where you want to actually have a bottleneck. After all, you don’t want users pushing to environments without your process being followed.

#### How long does it take to deploy?

If you are manually copying files between one or many servers, this is something you’ll want to address. Deploying a site should be easy, and repeatable. Look for opportunities to automate steps. Not only will this speed up deployment, but will also result make them more consistent.

#### How often do you deploy to Production?

Deploying early and often will limit the scope of changes per release, reduce issues in source control merging, speed up testing, decrease stress across your team by avoiding a monolithic “release day”, and be more responsive to your business’s needs.

## Designing your DevOps Pipeline

A DevOps pipeline is a set of (often automated) workflows, tools and gated processes around development, testing, releases to help your team be more productive, ultimately enabling you to develop and deploy faster.

When designing your pipeline, take into account any bottlenecks you have identified in your current development cycle, establish what you are trying to achieve and set your measures of success. For example, if your goal is to speed up the time it takes to QA your website by using more automated tools, your measure of success could be to reduce QA time by 25%. Shooting for a *finish line* will help you keep your team focused and ensure your efforts have quantifiable results.

### Your Build Pipeline - Checking in your Code

*Continuous Integration* is the practice of developers merging their changes back to the main source branch as often as possible. The sooner developers check their code into the master branch, the better. This allows code to be tested earlier and keeps each developer’s local repository closer to the code base that is currently deployed. This reduces friction during code merges and allows for automated tests to be run as often as they need to.

In order to maintain the master branch, ensure developers cannot check-in code directly to the master without the changes being approved. This will allow you to establish and control standards, provide teachable moments and promote knowledge sharing.

A successful merge will then kick-off a build process to validate the change.

### Your Build Pipeline - Validating the Code

Part of *Continuous Integration*, when the code is successfully merged into master, a build process is started. A build process will typically download the master source code, resolve and restore any external package dependencies, build the project, and run tests. Implementing a repeatable, automated build ensures that each and every code update is tested in a consistent way.

Automated Tests are a great way to test features and functionality and can drastically reduce manual QA testing time and limit bugs found in production. They allow you to identify bugs before they even get deployed.

If the build or any tests fail, the entire build process fails and the developer gets instant feedback on their build. At this point, the master branch’s build is broken and resources can be diverted to resolve this, submit a code change, and the build process is repeated until successful.

### Continuous Delivery vs Deployment

Once you have a successful build, you’ll want to deploy that to an environment. You can do this using a release pipeline.

*Continuous Delivery* is the practice of automatically releasing software using a validated build. This could be daily/nightly, weekly, monthly, or based on any other schedule. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

*Continous Deployment* is the practice of automatically releasing software using a validated build, but contrary to *Continuous Delivery*, this build is automatically deployed to without any manual intervention and does not follow a specific schedule. If you’ve managed to remove the need for manual QA or stakeholder approvals and are able to move to this model, you’ll be able to deploy multiple changes daily. Without a release day, this is an excellent way to accelerate the feedback loop with your customers and take the pressure off your team. Developers can focus on building software and see their changes live within minutes.

### Release Pipeline

Where possible, we recommend enabling *Continous Deployment* in your release pipeline. That way, after your build has been validated, deployment is made immediately to your QA/UAT environment without any human intervention. This saves time and ensures each new change kicks off its own release.

The first thing that should happen in this pipeline is to deploy your build to QA/UAT.

After your web app have been deployed to QA/UAT, consider having a manual approval process in your pipeline before the release is green-lighted for deployment to production. This allows for things like manual QA tasks or stakeholder review/approvals.

Once the release is approved of production, the deployment to production should be automatic and will conclude the last step in the release process.

Every code change should follow this process so that any build can be released with the click of a button.

## Recommended Tools

* **Source Control**: GitHub, GitLab, Azure DevOps, Atlassian
* **CI/D Tools**: Azure DevOps, GitLab, Atlassian, Jenkins
* **IDE**: VS Code, Visual Studio, Sublime Text