Feature Toggle and Dark Releases
Leveraging recent deployment trends to prepare for holiday traffic

Author: Manuel Alvarez, Enterprise Architect, Global Consulting Services, Akamai
Problem Statement

When developers gather around a campfire or meet at the bar to talk about horror stories, they all share a similar experience: The code worked in your local machine/dev environment but crashed the day of the release to production. These stories include plot twists such as “we did not have a way to test in production” and “we spent 72 hours working without sleep to get the code ready the very same day and we did not have time to fully test it.”

In the second situation, not having enough time to test, modern managers (product manager, scrum masters, or whoever is overseeing timelines) set deadlines well ahead of the release date. In some cases, however, the code or functionality is not ready and you must decide whether to not release the feature and remove the code from the master branch, do a partially working release (something that addresses some of the sprint stories), or hide the functionality so the code reaches production but cannot be executed.

The last option ties into a second problem: situations where the code is ready but cannot be tested or released in a production environment yet. The solution is to do a dark release where hidden code or functionality is released but can only be activated using a secret method, e.g., using an authentication cookie known by developers. There are other approaches, such as blue/green development where both environments are production ready, or using Infrastructure as a Service where you can spin off a production environment for testing. But I am a firm believer that there is no substitute for testing in production, even with wonders of spinning off copies of production environments. Why? Because users are unpredictable but your testing scripts and test cases normally aren’t.

Dark Releases and Feature Toggles

In any case, delivering code early should not be a BHAG (Big Hairy Audacious Goal) but rather the norm. In the words of Jez Humble,

“You can — and should — deploy your software to its production environment before you make it available to users, so that you can perform smoke testing and any other tasks such as waiting for caches to warm up. The job of smoke tests is to make sure that the deployment was successful, and in particular to test that the configuration settings (such as database connection strings) for the production environment are correct.”

The magic behind a dark release is the ability to release functionality or modify system behavior without changing application code. This technique is known as Feature Toggle or Feature Flag, as it allows you to toggle between a dark release (where nobody has access to the new functionality) and live release.

Challenges with Feature Toggle

The key behind feature toggle is that only the right “authority” must have access to it. This authority could be a developer, approval manager, external entity, or automated condition (e.g., a date/time match).

As an example, the Tesla Model 3 comes with all the hardware required to use enhanced autopilot (and the upcoming full self-driving). The feature can be enabled with an over-the-air software download, which means you could call in, use your credit card, and get the feature without leaving your home.

I do not have insight into Tesla engineering, but they have two options to ship the code that enables enhanced autopilot: It could indeed require a full software download (potentially a few GB) to the car, or they could ship the code as they do with the hardware while using a feature toggle to keep it off. You would think that the latter approach makes more sense, as that retains code uniformity (no software version WITH and WITHOUT to maintain) and does not require additional downloads; however, it presents new challenges.
Feature Toggle and Dark Releases: Leveraging recent deployment trends to prepare for holiday traffic

- How do you keep the toggle secure so it is not tampered with (hacked)?
- Access to enable this feature must meet the famous security AAA: Authentication, Authorization, and Accounting.
- There must be a proper governance to prevent the toggle from being deleted or deactivated by mistake; e.g., including test cases in your regression tests.
- Simplicity and collaboration across business units. In this example, the approval entity (support or sales person on the phone) is in a different department and might not even be technical; therefore, the toggle changes must be done using a user-friendly interface known by the approvers.

On top of these challenges, there is the concern of decentralized implementations, dirty code, and cascading feature toggles. Toggles are easy to implement in code, but hard to maintain. You will not know what is “enabled” if you do not come back and delete unused toggles after features are live.

Feature flags are common in CI/CD schemes where features are constantly deployed but not necessarily “released” in production. They are also very useful in web events and marketing campaigns as you can release the assets weeks in advance and schedule the toggle to make them live.

**Feature Toggle at your CDN**

CDNs front-face your app or web property and can centralize decisions. Detailed grant control and automation tools (e.g., API support, date matches, etc.) empower you to implement controls and governance around feature toggle. I will use the names that Akamai gives to features. Check your CDN capabilities and offerings to find the right name in their ecosystem.

*Why would I want to do a dark release of metadata instead of going live with the rest of my code release?*

- **Reduce risk:** A dark release allows you to test in a production environment without affecting customers.
- **Easy to implement:** Feature toggles are not always easy to implement. CDNs provide you with the ability to match almost anything in the incoming HTTP requests or client characteristics (device, location, etc.), thus letting you define complex and unique matches without changing your code/infrastructure.
- **Integrability:** APIs allow you to integrate the toggle control and release with your CI/CD tool.
- **Control:** Role and user permission controls who has access to modify configurations or make changes such as enabling feature toggles. You can limit or open grants to fit your organization’s needs.
- **Visibility:** New functionality is always launched with high hopes — and fear that something might not work as expected. You can add a new Content Provider Code under the same match as your feature to track traffic and get reports targeted. Another benefit of a new CP Code is that you can set up origin alerts that will notify you about errors such as timeouts or error responses reaching a defined threshold.

**How to do a Dark Release at your CDN**

There are several ways to leverage your CDN for feature toggle:

**New URL Paths or APIs**

CDNs can act as a reverse proxy and serve alternate content for a request. If the feature you are releasing lives in a new URL path, the best option is to send the request for the new URL to an alternate location, like a marketing page advertising the event, while only allowing access to select few (developers, testers, etc.). With Akamai, this feature is called Visitor Prioritization (VP) or API Prioritization, and it will effectively hide the URL path until you are ready to release.

Some advantages of using this feature are:

- Changes propagate almost instantaneously
- Give you granular control over how many users have access to the feature (e.g., you could enable the feature under a cookie match, for certain geos, or do a random lottery)
• Its powerful APIs provide a way to automate and integrate with your own environment
• The waiting room — the response clients receive when the toggle is off — provides a variety of options 
  (e.g., redirect to the API catalog or serve as an attractive marketing page)

The following diagram shows the flow and control:

You could also use any of the options presented below under Embedded Features, as they apply to your use case as well.

**Embedded Features**

Features must be matched in the metadata (CDN configuration) if you cannot match on URL patterns. In some cases, 
the new feature also requires metadata changes. For example, migration to a new CMS or technology stack most likely 
requires an update of the origin settings in your configuration.

You can architect your configuration and dark release in different ways. A common approach is to use a variable that 
sets the behavior ON and OFF. The variable will have a default value of FALSE. Developers, testers, and other people you 
want to grant access can bypass the control by using a secret cookie, IP ACL, or Edge authentication. Naturally, releasing 
the feature means switching the variable to ON. As with most architectural designs that belong to your organization, be 
sure to document the feature, how it works, and how it can be modified. You do not want to rely on key personnel or 
word of mouth to support and maintain your dark releases.

An approach that is gaining popularity is to leverage existing CDN capabilities that have standard support and are 
well documented; in other words, use products rather than custom implementations. Customers using blue/green 
development switch “origins” to release new code. CDNs have capabilities to change the origin instantaneously using 
APIs or an interface. The trick is that you can define both the blue and green origin to be the same DNS endpoint while 
leveraging the CDN origin split capabilities to work as the feature toggle. Your new development is implemented under 
the origin that does not have traffic, to prevent users from seeing the release ahead of time. With Akamai, this feature 
is called Phased Release Cloudlet, and the new release feature can be implemented inside the Cloudlet Origin to ensure 
the right portion of traffic is applied.

The image below exemplifies the flow with this option. Phased Release is controlled by developers or operations using 
APIs that could be integrated with your preferred architecture management or deployment tool.

Whichever approach you prefer, here are some best practices:

1. Use a meaningful name for variables and/or origins to reduce the risk of confusion and help you clean up. 
   For example, “new” and “old” become really confusing when you are doing blue/green development.

2. It is strongly recommended after a successful release to clean your code and remove the feature toggles that will 
   remain ON. It is common to have multiple toggles in a release that, combined with past release toggles, make it hard 
   to understand what is currently live. A simple clean-up can go a long way toward avoiding misunderstandings.
3. Granting access to your internal team using a cookie is the easiest option. You could even set an internal welcome page that sets the cookie, thus allowing QA, support, and business partners to get access without asking them to manually add a cookie — something many managers might not know how to do.

Note: Cookies are not a secure mechanism and are not recommended for scenarios where the risk of the cookie details leaking to the world could have significant impact for the business or your operation.

4. For high-visibility events, flip the switch slowly by granting access by geo region, random match, customer category (e.g., premium users), or device (e.g., desktop users).

5. Always be ready to roll back. Accidents and bugs happen; hence, you must always be ready to revert.

6. Use fast activation or a remote control to enable the feature. Fast activation allows the activation of a new version with changes in the switch value in minutes. A remote control allows you to make changes instantaneously.

7. Automate the process. Integrate the CDN control with your release tools and processes. Retain, however, the ability to override the automated process. This could be as simple as documenting the APIs you use to make changes to the feature toggle or as complex as coding a kill switch.

**Designing with Akamai**

A mature design pattern could be to release new features under a new endpoint (URL) and use Akamai to proxy the old, SEO-friendly path to the new feature path. Using variables, as described above, you can control who gets access to the new feature, and when.

We will be glad to assist you if you have further questions or would like to discuss the best option for your use case. Please reach out to your Akamai account team or email consulting@akamai.com.