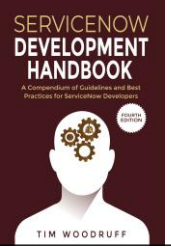


CLONING

Cloning from your production environment over sub-prod environments like Dev or Test, is a very important step for any ServiceNow customer. In this section, we're going to learn some important best-practices for cloning, and discuss some suggestions for clone cadences (the frequency and process with which to clone your environments to minimize both instance drift, and down-time).

ServiceNow now lets administrators manage instance cloning much more fully from *within* your instance, using the **System Clone** application navigator modules[1]. Their documentation (see footnote) is actually quite good on this topic, so I won't repeat what they say here, but I will point out a few important notes to be aware of.

First off, it's important to be aware of the fact that clones copy data from the **most recent platform backup of the source instance,** which typically happens during an off-peak maintenance window. So let's say you have a clone scheduled for Friday at Noon, but someone made or deployed a change to production on Friday morning (*boo, hiss!*). That change may not be captured in the clone, and you would end up with an already-drifted sub-prod instance!

It's also a good idea to keep in mind that if you try to clone *from* an instance running one version of ServiceNow, *over* an instance running another version, a central web service will automatically modify the target instance to match the version of ServiceNow that the source instance is running. This is normally desirable, but it should be noted that this process may take **up to 8 hours** to complete on top of the normal clone time! This process starts 8 hours **before** the scheduled clone time, so be sure to have any work or records you need backed up exported from the target instance at least 8 hours before the scheduled time of your clone!

CLONE CADENCES

The increasing differences between your production environment and sub-prod environments, is called **instance drift**. Instance drift is **bad**. The way to minimize instance drift, is to clone over your sub-prod instances (from production) on a regular basis. Cloning is one of the most important things you can do to ensure a clean deploy, healthy sub-prod environments, and to minimize the risk of carrying over artefacts of old or abandoned development.

It's important to have a well-defined clone cadence in order to prevent these issues. Among other things, your organization's clone cadence might be different depending on how many sub-prod instances you have. In this section, I'll outline some suggestions for potential clone cadences, depending on how many instances you have.

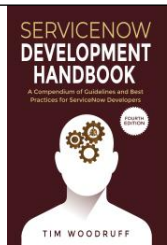
***Note:** Cloning wipes all data in the target environment, including any active development work or test configurations. Be sure to export any data or code that hasn't yet been captured in Update Sets or Applications. The "Preserve Data" module lets you exclude specific tables or rows from being overwritten – use it wisely for tables like sys_email or sys_properties.*

Multi-Dev Clone Cadence

We'll start with what, in *my opinion*, is the optimal setup from a cost-vs-benefit perspective: **one production environment** (obviously), **one test environment**, and **two development environments** (*with potentially a third if you have a large "citizen developer" population, so they can bork that environment without impacting your team's work*). The two dev environments can be used alternately, switching from one to the other each sprint, and cloning each environment every *other* sprint.

This setup has several advantages:

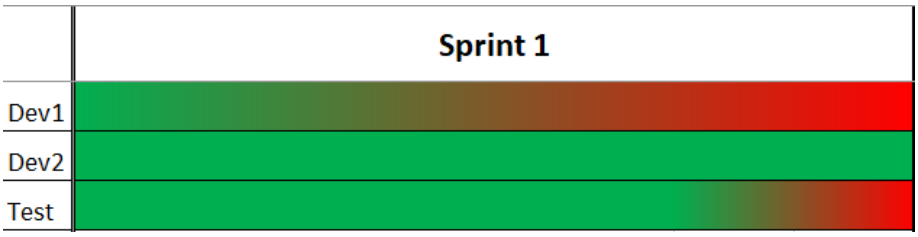
[1] *You can find complete documentation on ServiceNow's cloning setup and configuration process, here: https://hbk-cloning-docs.snc.guru*



1. Build new work for the current sprint in one environment, while the previous environment is retained (along with all work done in it).
    a. This is especially valuable if something was not captured in the proper Update Set, or was not captured at all (such as if someone forgot to push an important data record or the current version of a Scheduled Script Execution into their Update Set).
2. If a bug is identified in the previous sprint's work after the new sprint has started, the environment in which it was built still remains pristine. This allows you to fix that bug in the environment in which it was created, without risking carrying over work from the new sprint that isn't ready to be deployed yet.
    a. For example, imagine if in sprint 10 you built something involving a Script Include. After sprint 11 starts, a bug is identified that requires that that Script Include be modified. However, that SI is also already being modified for an enhancement to be included in sprint 11; but it isn't ready yet. If you had only a single dev environment, fixing this bug in a timely manner would be quite difficult! But, as long as you remember to apply the "fix" to the code in the environment that sprint 11 is being built in as well, it's much easier with two dev environments.
3. Allows clones to happen at the end of a sprint, rather than between sprints. This means less down-time while you wait for a clone to finish, before work can resume. This also allows you to do a clone between **every sprint**, ensuring the minimum possible **instance drift**.

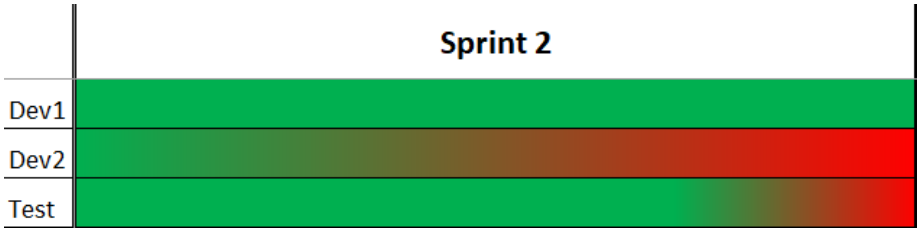
To help you visualize instance drift as a function of time and the work being done in an instance, and to visualize the clone cadence I recommend with this setup, here is a step-by-step guide, with visualizations. Instance health will be represented by color (green being healthy, red indicating a greater quantity of "drift" from production) over time, from left to right.

To begin, let's start Sprint 1 in our Dev1 environment. Note that this means odd-numbered sprints will always be built in Dev1, and even-numbered sprints will always be done in Dev2. Here is a representation of our instance drift over time:



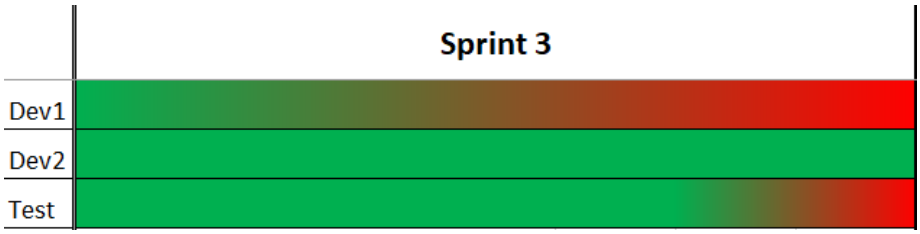
Note that Dev1 increases in drift over time throughout the sprint, but the Test environment only drifts at the end of the sprint, when work from Dev1 begins being deployed to it.

At the end of Sprint 1, we deploy Sprint 1's Update Sets to Test and then (after testing), to *both* Production, and Dev2. Afterward, we begin work on **Sprint 2** in **Dev2**.



Throughout Sprint 2, Dev2 drifts as work is done in it. Simultaneously, testing may be continuing in the Test environment, and fixes may be done in the Dev1 environment. At some point **during Sprint 2**, after Sprint 1 is deployed to production, the **Test and Dev1 environments** will be **cloned over**, in preparation for Sprint 3.

At the end of Sprint 2, we deploy Sprint 2's Update Sets to Test, and then to Production and the recently cloned Dev1 in preparation for Sprint 3.



From here on, we just lather, rinse, and repeat the same process, alternating between working in Dev1 for odd-numbered sprints, and Dev2 for even-numbered sprints.

Single-Dev Clone Cadence

Working with a single Dev instance is slightly riskier and may lead to additional instance drift between clones (to avoid down-time between sprints) but can save some money and lends itself to an overall simpler approach to instance cloning. For a single dev environment, you've just got to choose how often you want to clone and stick to it.

Organizations with a single development environment tend to clone less frequently than those with multiple dev environments, but the trade-off for cloning less frequently (and thus suffering additional instance drift and associated risk) is that you get to avoid some down-time between sprints. Rather than waiting for a sprint to be complete, then waiting for testing to complete, waiting for production deployment, waiting for smoke testing, then scheduling a clone and waiting for that to finish as well before being able to begin each sprint, you may simply want to clone once per quarter, or once every 2-3 sprints for example.

If you clone only once every 3 sprints, that cuts your down-time down by 2/3rds as compared with cloning after each sprint. It's still not as efficient as having two development environments, but if you have a small dev team and don't want to spend the extra money for that second dev environment, this is a perfectly acceptable solution. It does, however, require a bit of additional diligence when it comes to being mindful of instance drift between clones, and being cognizant of what is and isn't yet in production (especially when doing bug-fixes for previous sprints).