The Terrible Twos - Part 1: Your Web App is Ready to Grow

=====================

Chris Cameron, Spiria

# Background

Look around the web and you'll find countless tutorials about how you can make a Twitter clone in three lines of code using today's latest web framework. ToDo apps are so plenty these days there's [a website](http://todomvc.com/) which tracks over 60 basic implementations across JavaScript frameworks.

Very quickly, however, these basic implementations suffer when web traffic increases. Often overlooked among the positive progress in web frameworks and their sample applications is the ability to handle rising demand for their services.

Once an application reaches this first critical point in traffic it has reached The Terrible Twos.

# The Terrible Twos

You've probably heard about the terrible twos as it applies to kids. When children reach two years old, they start to assert independence from their parents, cycling back to dependence when they are overwhelmed. Typical symptoms of this phase in a child's life include a lot of exhaustion and misbehaviour.

I think web applications go through a stage like this. They still rely on their "parents" (dev and ops) for provisioning, reboots, resource increases, and other things the application cannot do on its own. Health checks? Mom and dad listen to the baby monitor - dev and ops check `top` via ssh. Toddler/app fell over? Once you hear the cries (from customers) you come rushing over to put them/it back up. Dirty diaper... never mind.

How do you know when your application reaches the terrible twos? If you are tracking your web application's traffic rate (and you should be!), you may be looking at a metric called Requests Per Second (rps), or for lower traffic applications you may be tracking Requests Per Minute (rpm).

In the web applications that I've seen, I think the terrible twos start around 120rpm, or 2 rps (Aha, TWO! Perfect narrative synergy).

# Rising Expectations

Just like toddlers, applications around 2 rps begin to receive additional expectations. Every minute your application needs to provide a response to at least 120 requests. Naturally, you'll want to be able to respond in (far) less than half a second or else requests will begin to queue. Requests don't arrive exactly every 500ms so there is always \*some\* queuing. Once the queue fills up, a single threaded application will start to experience poor behaviour.

Next, let's examine some common kinds of misbehaviour.

# Exhaustion

Resource exhaustion when using a basic application stack can happen regularly around 2 rps.

The first thing you might notice is your request pool filling up, leading to dropped requests, and (most likely) errors in your framework that rhyme with `RequestTimeout`. You might be tempted to increase the size of your request pool: don't. You'll experience more traffic gains soon enough and be right back to where you were.

You may also see instances where your RAM or disk fill up, causing slowness, and potentially crashes. Especially if you're just starting out you probably chose a fixed set of resources for your application, perhaps the basic $5/month droplet on Digital Ocean or a standard tier Heroku dyno. This doesn't mean that you need to increase your RAM or Disk space right away. Although it can be a useful bandaid while you investigate, increasing RAM and Disk size will cost more than your web traffic or infrastructure budget justifies.

Sometimes your (virtual) CPU seems pinned at 100%. What ,if anything, you do about this will depend on the cause of the CPU usage. You don't want to over-provision your CPU because you will incur the full costs even if you only use a fraction of the processing time. It may make sense to increase the number of CPUs, tier of CPU, or even the isolation (virtual->dedicated) of CPU, but at this stage there are usually better and cheaper solutions.

Remember: making full use of your resources - Request Pool, CPU, RAM, or Disk - is not a problem by itself. It is the correlation to poor behaviour that makes it a problem we need to solve.

# Misbehaviour

What are some of the things a toddler starts to do around two?

- They refuse to start a new task.

- They take a long time to accomplish a task.

- They say they'll do something but then don't. Occasionally they'll agree to a whole series of tasks they won't actually do.

- They refuse to stop a task.

- They stop a task abruptly and leave an absolute mess for you to clean up.

Sound familiar? You'll start to see this behaviour in your application as well. Applications that can't boot up, or are constantly blocked from achieving anything. Request pool full and requests get abandoned by clients. Requests that take phenomenally long times without actually completing for the clients that hang on.

Eventually this misbehaviour results in application and/or dependency crashes and request timeouts which can leave state corrupt or missing. These "Tantrums" often result in manual intervention by the parents, err, developers and operations people who have to reboot the app (or machine!), clear or warm cache(s), and watch to make sure the system goes back to a point of stability.

# Next

In this article we discussed some of the symptoms of the terrible twos. In the coming articles we'll investigate some techniques for dealing with the rising expectations being placed on your application by reducing resource use, delegating work to specialized workers, storing data in caches, and doing some very basic horizontal scaling.

In other words, we're going to get your toddler to make some new friends and then teach it some discipline too :)