Approaches for application request throttling

Maarten Balliauw
@maartenballiauw

Who am I?

Maarten Balliauw
Antwerp, Belgium
Developer Advocate, JetBrains
Founder, MyGet
AZUG
Focus on web
- ASP.NET MVC, Azure, SignalR, …
- Former MVP Azure & ASPInsider

Big passion: Azure
http://blog.maartenballiauw.be
@maartenballiauw

Agenda

Users and traffic patterns

Rate limiting and considerations

- Which resources?
- Which limits?
- Who to limit? Who not to limit?
- What when a limit is reached?
- Where to limit?


Users…

MyGet

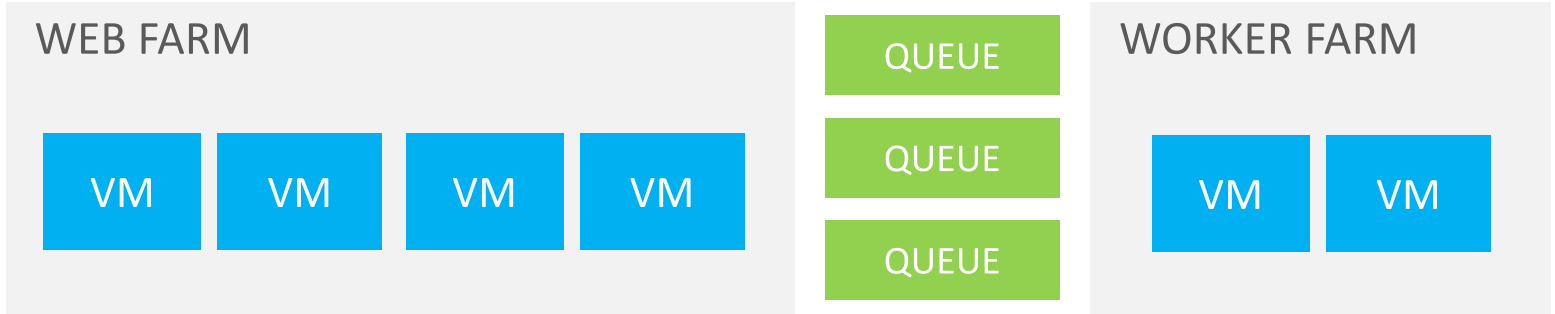
Hosted private package repository – www.myget.org
- NuGet, NPM, Bower, Maven, VSIX, PHP Composer, Symbols, …

HTTP-based
- Web UI for managing things
- API for various package managers
  - *PUT/POST – Upload package*
  - *DELETE – Delete package via API*
  - *GET – Fetch metadata or binary*

We're using background workers

Example: package upload

- PUT/POST binary and metadata to front-end
- **`PackageAddedEvent`** on queue with many handlers handled on back-end
  - *`ProcessSymbols`*
  - *`UpdateLatestVersion`*
  - *`Indexing`*
  - …



What could possibly go wrong…

Too many uploads incoming!

- Front-end
  - *IIS server needs workers to read the incoming network stream*
  - *Application logic has to check credentials, subscription, quota*
- Back-end
  - *Delays in queue processing (luckily workers can process at their own pace)*

Too many uploads that are too slow!

- Front-end
  - *IIS server needs lots of workers to slowly copy from the network stream*
  - *Workers == threads == memory == synchronization == not a happy place*

What could possibly go wrong…

Too many downloads!

- Application logic has to check credentials, subscription, quota
- 404's still need that application logic…
- Package managers are crazy!

		Total # requests	Total # 404's	% 404's
# of packages in solution	200	800	600	
# on NuGet.org	190	200	10	5%
# on MyGet feed 1	5	200	195	97,5%
# on MyGet feed 2	4	200	196	98%
# on company-internal TeamCity	1	200	199	99,5%

Other examples

Web UI requests

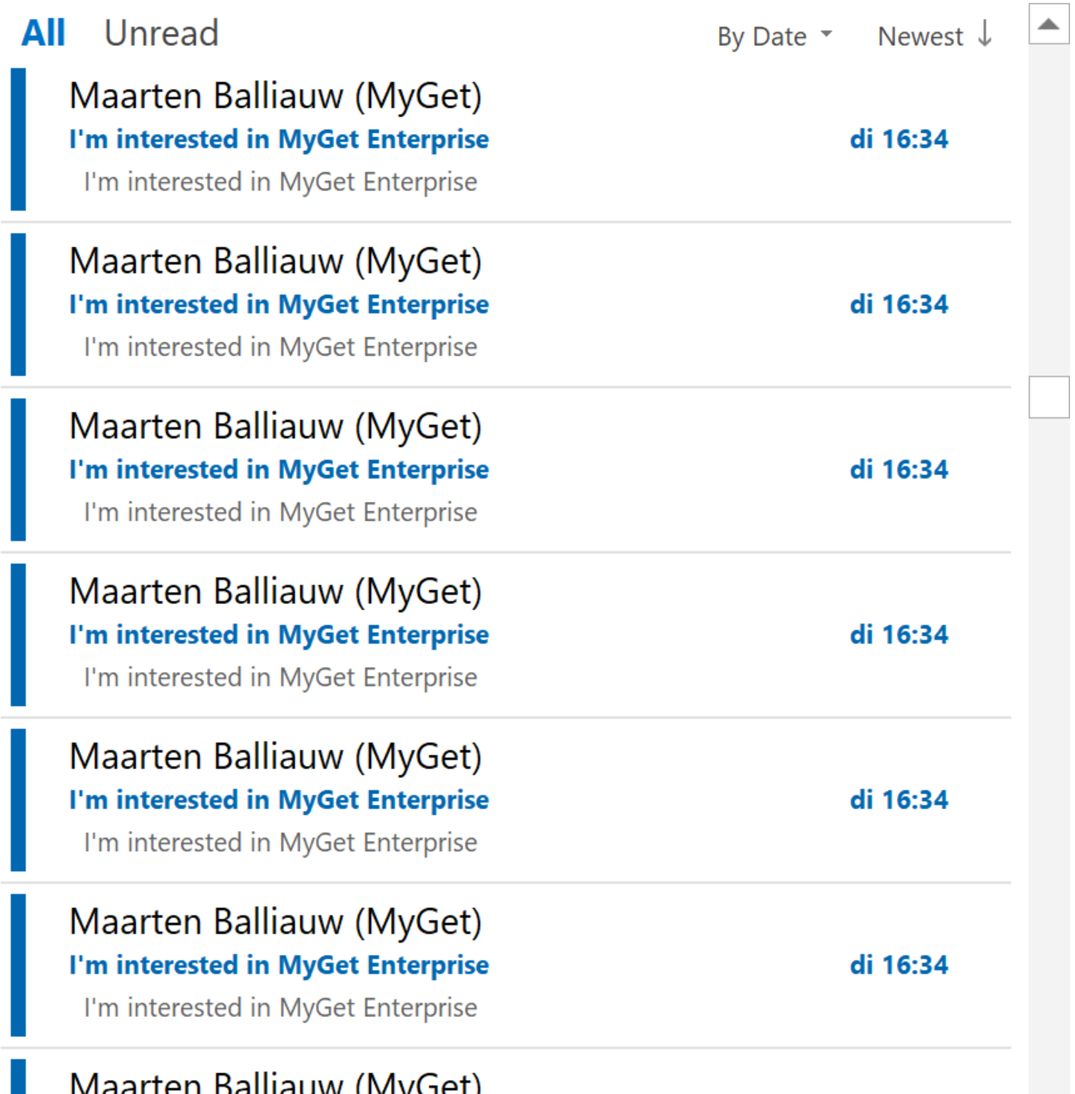
- Trying to register spam accounts
- Trying to brute-force login/password reset
- Trying to validate credit card numbers via a form on your site

Robots / Crawlers

- Imagine a spider adding 20k items to a shopping cart
- For us, usually fine (e.g. Googlebot by default *up to* 5 req/sec)
- Limiting is easy with **`rel="nofollow"`** and **`robots.txt crawl-delay`**

Recent, real-life example

What the?!? 610156



Rate limiting!

(or “throttling”)

Rate limiting – what?

Limits # of requests in a given timeframe

- Or limits bandwidth, or another resource – up to you

Helps eliminate:

- Unexpected traffic patterns
- Unwanted traffic patterns (e.g. script kiddie brute-force login)
- Potentiallly damaging traffic patterns
  (accidental and malicious)

Rate limit everything.

- Maarten Balliauw

Rate limiting – everything???

Everything that could slow down or break your application

Typically everything that depends on a scarce or external resource

- CPU
- Memory
- Disk I/O
- Database
- External API

So yes, everything…

Let's do this!

Database with table **`Events`**

- **`UserIdentifier`** – who do we limit
- **`ActionIdentifier`** – what do we limit
- **`When`** – event timestamp so we can apply a query

Filter attribute

```
SELECT COUNT(*) FROM Events WHERE UserIdentifier = <user> AND
ActionIdentifier = <action> AND When >= NOW() - X

INSERT INTO Events (<user>, <action>, NOW())

DELETE FROM Events WHERE UserIdentifier = <user> AND
ActionIdentifier = <action> AND When < NOW() - X
```

Let’s do this!

demo

Rate measuring

That database was a bad idea!

Very flexible in defining various limits or doing combinations

Very flexible in changing limits, e.g. changing the time period

The database will suffer at scale…

- Every request is at least 2 – 3 queries
- Constant index churn
- We need to manually run DELETE to remove old events
- Database size!

That database was a bad idea!

We created a denial of service opportunity!

- SELECT, INSERT, DELETE for every request

Consider a simpler technique to limit # of operations

- Ideally just a simple counter
- “Buckets”

Quantized buckets

Create “buckets” per **<identifier>** and **<timespan>**

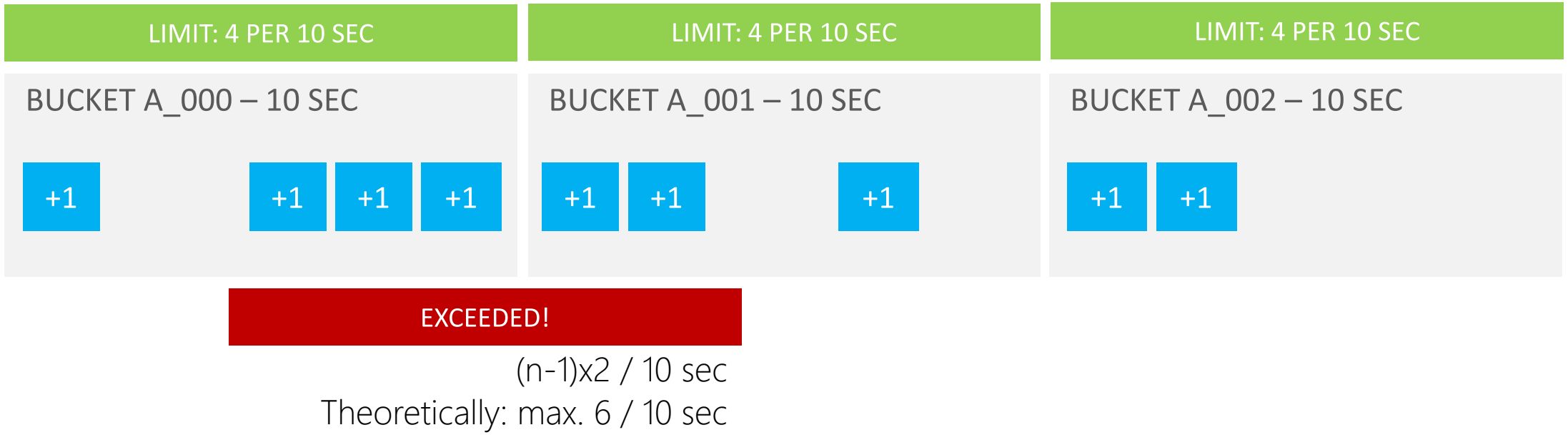
Use **incr <bucket>** on Redis and get back the current count per **<timespan>**

```
public string GetBucketName(string operation, TimeSpan timespan)
{
    var bucket = Math.Floor(
        DateTime.UtcNow.Ticks / timespan.TotalMilliseconds / 10000);

    return $"{operation}_{bucket}";
}

Console.WriteLine(GetBucketName("someaction", TimeSpan.FromMinutes(10)));
// someaction_106062120 <-- this will be the key for +/- 10 minutes
```

Quantized buckets

Super easy and super cheap (atomic write and read on Redis, auto-expire LRU)

Not accurate… (but that may be ok)



Leaky bucket

"Imagine a bucket where water is poured in at the top and leaks from the bottom.

If the rate at which water is poured in exceeds the rate at which it leaks, the bucket overflows."

Widely used in telecommunications to deal with bandwidth/bursts.

Bucket algorithms

demo

Bucket approaches

QUANTIZED BUCKET

Create "buckets" per `<identifier>_<timespan>`

No rolling window (new bucket every `<timespan>`)

Simple key/value store is sufficient, 1 atomic read+write

Old keys can auto-expire (unlike our DB approach)

LEAKY BUCKET

Get `<delta>` tokens, with maximum `<count>` per `<timespan>`

Rolling window, smooths traffic, allows bursts when bucket has capacity

Need to store # tokens, last refill, concurrency needs to be taken into account

Or use a FIFO queue of timestamps (works great with Redis sorted set)

Redis sorted set as a bucket

Demo

Cool! That’s it, right?

Deciding on limits

Things to decide on

Decide on the resources to limit

Decide on a sensible limit

Come up with an identifier to limit on

Decide on exceptions to the rule

Which resources to limit?

…

Rate limit everything.

- Maarten Balliauw

What are sensible limits?

Approach 1

1. Figure out current # of requests for a certain resource
2. Set limits
3. Get angry phone calls from customers

Approach 2

1. Figure out current # of requests for a certain resource
2. Set limits, but only *log* when a request would be limited
3. Analyze logs, set new limits, …
4. Start rate limiting
5. Keep measuring

Will you allow bursts or not?

Laddering! Different buckets per identifier and resource…

10 requests per second can be 36000 requests per hour.

But 10 requests per second could also be 1000 requests per hour.

Bucket	Operation A	Operation B	Operation C
Per second	10	10	100
Per minute	60	60	500
Per hour	3600	600	500
…			
	Steady flow of max. 10/sec	Steady flow of max. 10/sec, but only 600/hour max.	Bursts of up to 100/sec, but only 500/hour max.

What will be the identifier?

Per IP address?

- But what with NAT/proxy?

Per user?

- But how do you limit anonymous users?

Per session?

- But what when the user starts a new session for every request?
- Or what if there is no such thing as a session?

Per browser?

- But everyone uses Chrome!

What will be the identifier?

Probably a combination!

- IP address (debatable)
- + User token (or “anonymous”)
- + Session token
- + Headers (user agent + accept-language + some cookie + …)

Decide on exceptions

Do we rate limit all users? Do we have separate limits for certain users?
- Dynamic limiting

Do we rate limit all IP addresses?
- What about ourselves?
- What about our monitoring tools?

What about web crawlers?

What about certain datacenter ranges? (https://github.com/client9/ipcat)
- “IP addresses that end web consumers should not be using"

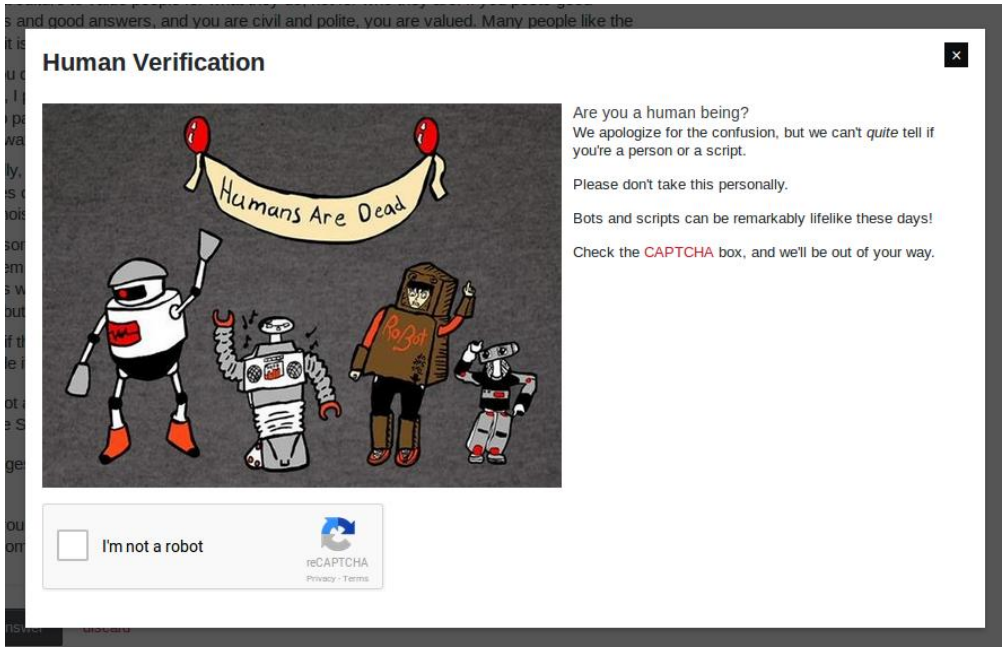
Responding to limits

What when the user hits the limit?

Do we just “black hole” and close the connection?

Do you tell the user?

- API: status code **`429 Too Many Requests`**
- Web: error page stating rate limit exceeded / captcha (StackOverflow)



Try to always tell the user

Format? Depends on **`Accept`** header (**`text/html`** vs. **`application/json`**)

Tell them why they were throttled

- Can be a simple link to API documentation

Tell them when to retry (e.g. GitHub does this even before rate limiting)

```
Status: 200 OK
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4999
X-RateLimit-Reset: 1372700873
```

Where do we limit?

Rate limiting – where?

MvcThrottle

*Runs as **action filter***

Requests per timespan

Per action, user, IP, … (so knows about actions)

Owin.Limits

*Runs as **OWIN middleware***

Bandwidth, concurrent requests, …

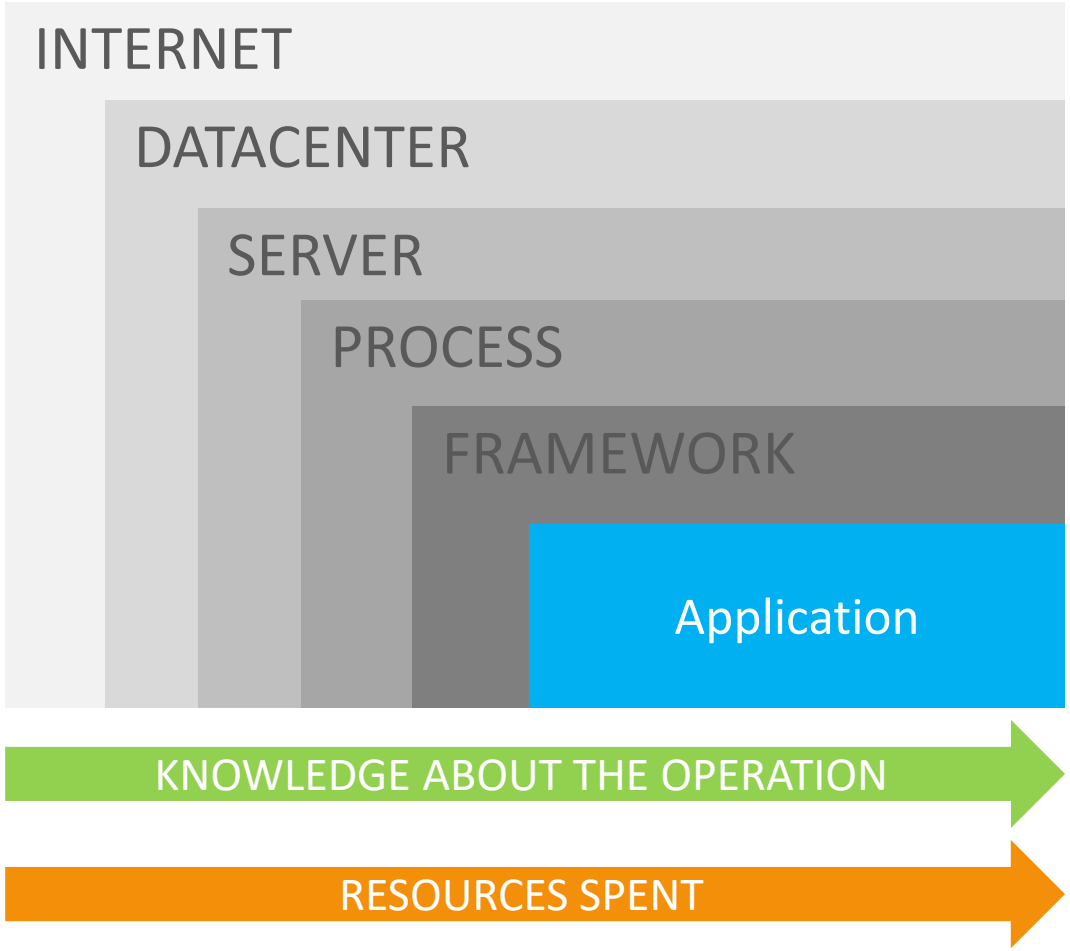
No knowledge about application specifics

Many, many others

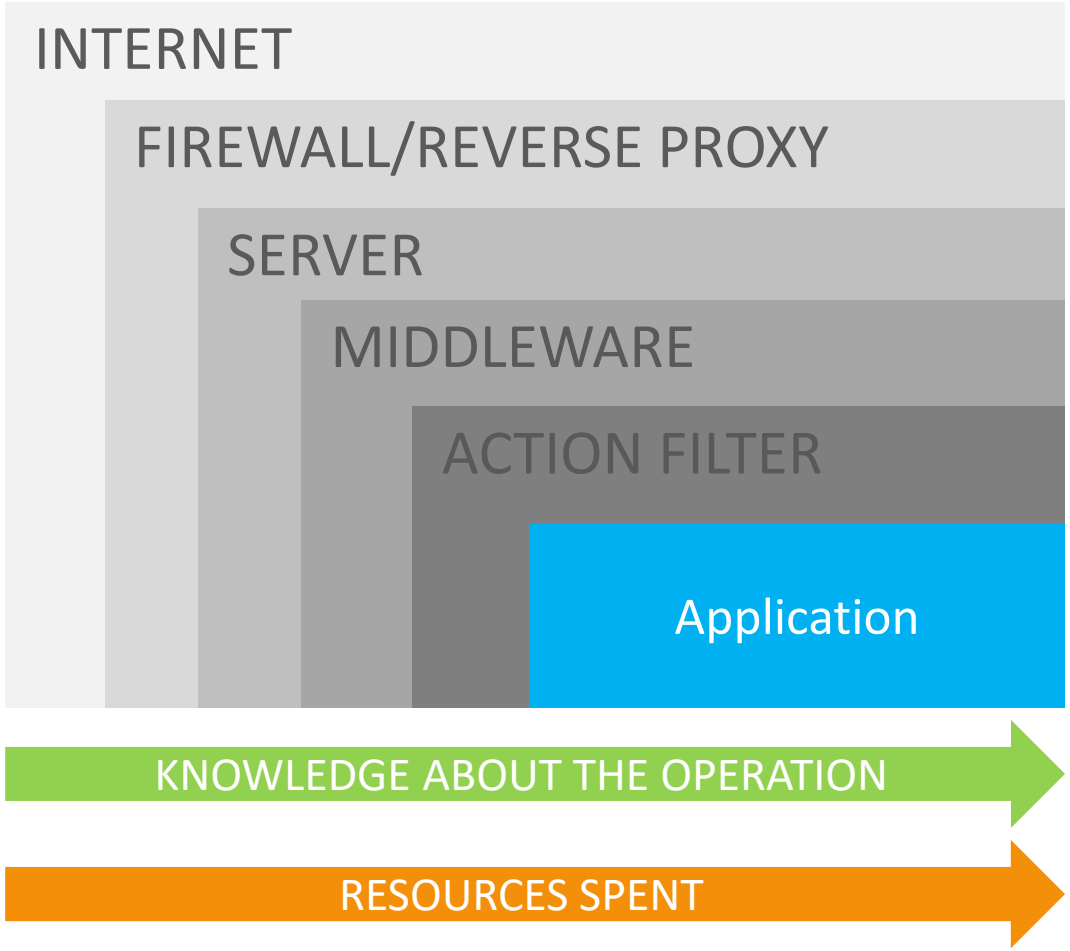
MvcThrottle

Demo

How far do we allow traffic before saying no?



How far do we allow traffic before saying no?



What options are there?

In our application

- `ActionFilter` / `Middleware` / `HttpModule` / …
  - Easy to add custom logic, based on request details

On the server

Outside of our server

Outside of our datacenter

What options are there?

In our application

On the server

- IIS has dynamic IP restrictions, bit rate throttling, `<limits />`
- Kestrel minimum speed throttle
- Found these less flexible in terms of configuraton…
  - *E.g. IIS dynamic IP restrictions returns* `403 Forbidden`*, wth!*
- Not a big fan, as these are usually HttpModules anyway (and thus hit our app)

Outside of our server

Outside of our datacenter

What options are there?

In our application

On the server

Outside of our server

- Reverse proxy (IIS Application Request Routing, NGinx, HAProxy, Squid, …)
- Traffic does not even hit our application server, yay!

Outside of our datacenter

Rate limiting with NGinx

Demo

What options are there?

In our application

On the server

Outside of our server

Outside of our datacenter

- Azure API management, CloudFlare
- Filters traffic very early in the request, yay!
- Often also handle DDoS attacks
- Often more expensive


IN MEMORIAM
MEN OF THIS BANK WHO GAVE
THEIR LIVES IN THE GREAT WAR
MCMXIV
MCMXVIII

Conclusion

Conclusion

Users are crazy! (typically unintended)

We need rate limiting

- Decide on the resources to limit (tip: everything)
- Decide on a sensible limit (tip: measure)
- Come up with an identifier to limit on
- Decide on exceptions
- What when the user reaches a limit?
- Decide where in the request/response flow to limit

Thank you!

http://blog.maartenballiauw.be
@maartenballiauw