Indexing and searching NuGet.org with Azure Functions and Search

Maarten Balliauw
@maartenballiauw

JET BRAINS

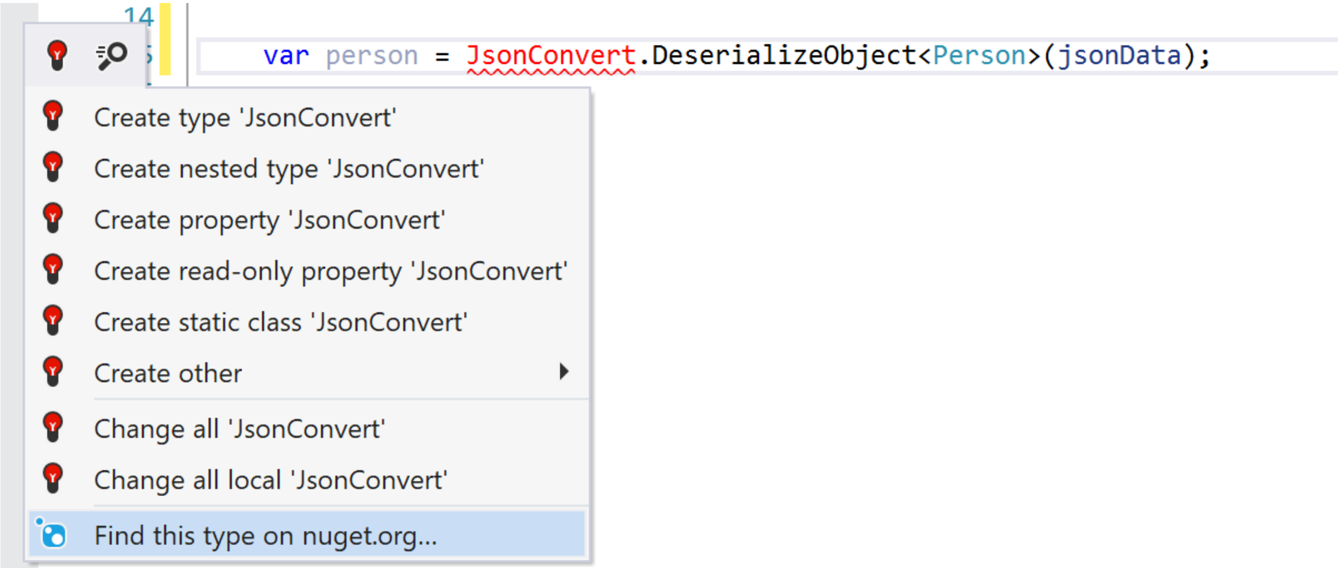
WAZUG.NL
AZURE USER GROUP NL

“Find this type on NuGet.org”

“Find this type on NuGet.org”

In ReSharper and Rider

Search for namespaces
& types that are not yet referenced



```
var person = JsonConvert.DeserializeObject<Person>(jsonData);
```

Create type 'JsonConvert'
Create nested type 'JsonConvert'
Create property 'JsonConvert'
Create read-only property 'JsonConvert'
Create static class 'JsonConvert'
Create other
Change all 'JsonConvert'
Change all local 'JsonConvert'
Find this type on nuget.org...

“Find this type on NuGet.org”

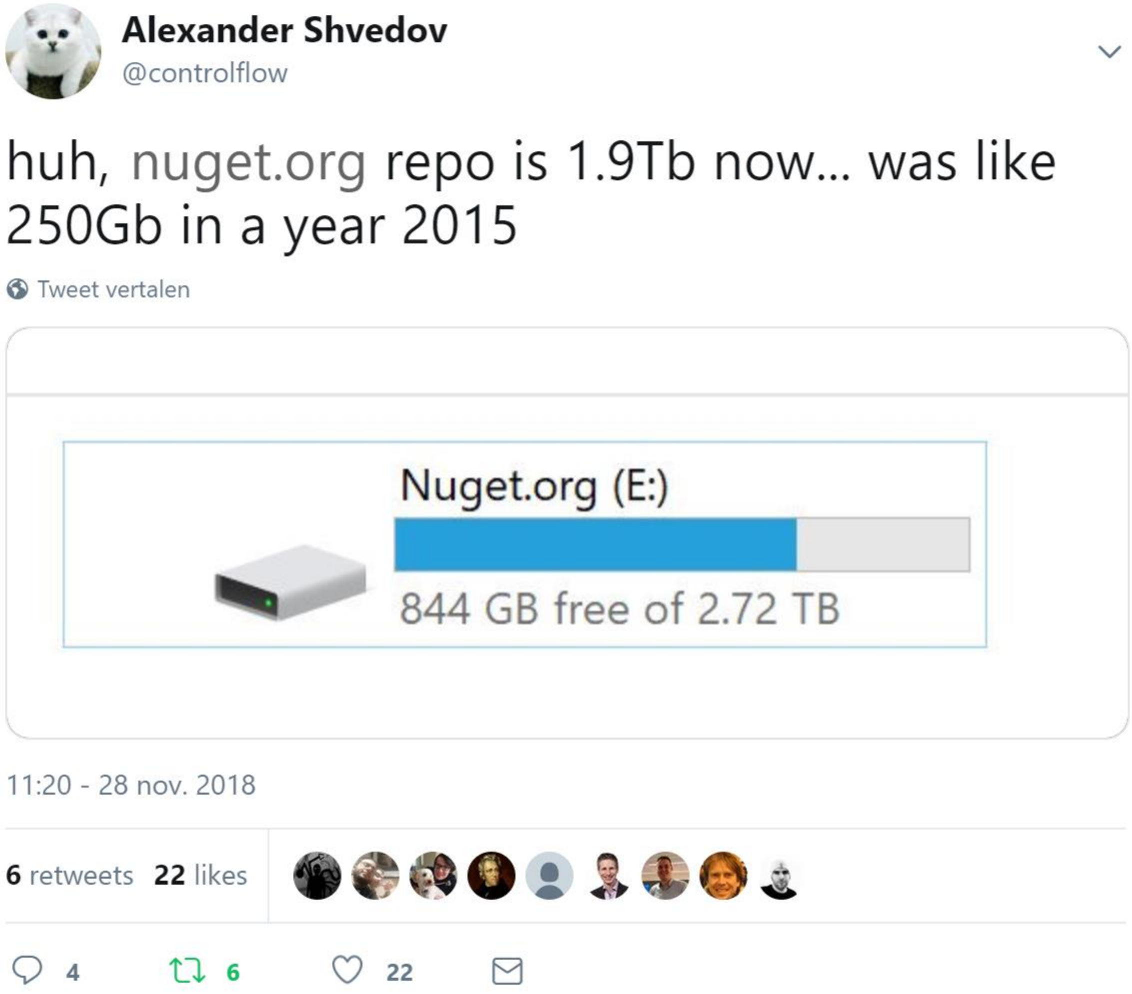
Idea in 2013, introduced in ReSharper 9

(2015 - https://www.jetbrains.com/resharper/whatsnew/whatsnew_9.html)

Consists of

- ReSharper functionality
- A service that indexes packages and powers search
  - *Azure Cloud Service (Web and Worker role)*
  - *Indexer uses NuGet OData feed*
    - https://www.nuget.org/api/v2/Packages?$select=Id,Version,NormalizedVersion,LastEdited,Published&$orderby=LastEdited%20desc&$filter=LastEdited%20gt%20datetime%272012-01-01%27

NuGet over time…



Alexander Shvedov
@controlflow

huh, nuget.org repo is 1.9Tb now… was like 250Gb in a year 2015

Tweet vertalen

11:20 - 28 nov. 2018

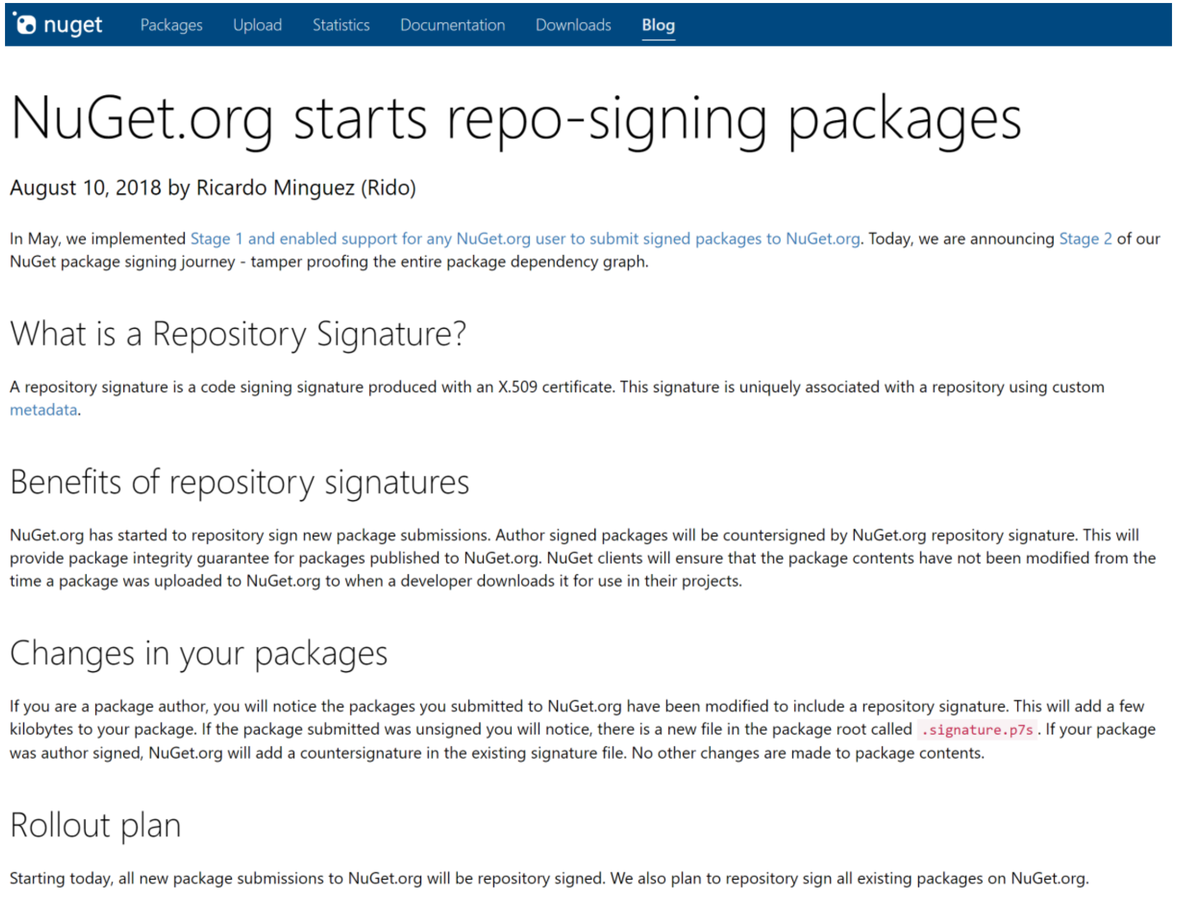
6 retweets **22** likes

4 6 22

https://twitter.com/controlflow/status/1067724815958777856

NuGet over time…

- Repo-signing announced August 10, 2018
- Big chunk of packages signed over holidays 2018/2019
- Re-download all metadata & binaries
  - Very slow over OData



NuGet.org starts repo-signing packages

August 10, 2018 by Ricardo Minguez (Rido)

In May, we implemented Stage 1 and enabled support for any NuGet.org user to submit signed packages to NuGet.org. Today, we are announcing Stage 2 of our NuGet package signing journey - tamper proofing the entire package dependency graph.

What is a Repository Signature?

A repository signature is a code signing signature produced with an X.509 certificate. This signature is uniquely associated with a repository using custom metadata.

Benefits of repository signatures

NuGet.org has started to repository sign new package submissions. Author signed packages will be countersigned by NuGet.org repository signature. This will provide package integrity guarantee for packages published to NuGet.org. NuGet clients will ensure that the package contents have not been modified from the time a package was uploaded to NuGet.org to when a developer downloads it for use in their projects.

Changes in your packages

If you are a package author, you will notice the packages you submitted to NuGet.org have been modified to include a repository signature. This will add a few kilobytes to your package. If the package submitted was unsigned you will notice, there is a new file in the package root called `.signature.p7s`. If your package was author signed, NuGet.org will add a countersignature in the existing signature file. No other changes are made to package contents.

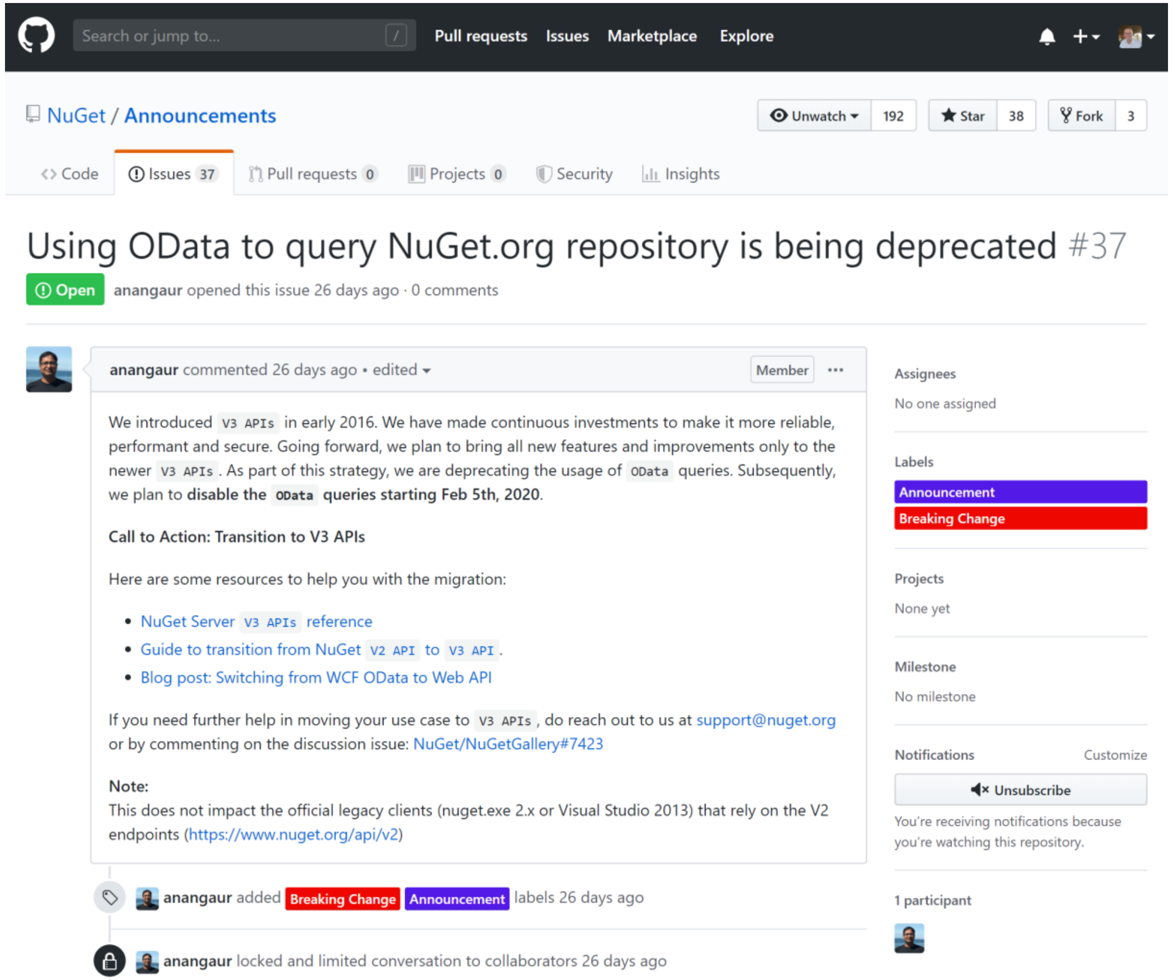
Rollout plan

Starting today, all new package submissions to NuGet.org will be repository signed. We also plan to repository sign all existing packages on NuGet.org.

https://blog.nuget.org/20180810/Introducing-Repository-Signatures.html

NuGet over time...

OData API being deprecated!



Using OData to query NuGet.org repository is being deprecated #37

Open anangaur opened this issue 26 days ago · 0 comments

anangaur commented 26 days ago • edited Member

We introduced `V3 APIs` in early 2016. We have made continuous investments to make it more reliable, performant and secure. Going forward, we plan to bring all new features and improvements only to the newer `V3 APIs`. As part of this strategy, we are deprecating the usage of `OData` queries. Subsequently, we plan to **disable the `OData` queries starting Feb 5th, 2020**.

Call to Action: Transition to V3 APIs

Here are some resources to help you with the migration:

- NuGet Server `V3 APIs` reference
- Guide to transition from NuGet `V2 API` to `V3 API`.
- Blog post: Switching from WCF OData to Web API

If you need further help in moving your use case to `V3 APIs`, do reach out to us at support@nuget.org or by commenting on the discussion issue: NuGet/NuGetGallery#7423

Note:
This does not impact the official legacy clients (nuget.exe 2.x or Visual Studio 2013) that rely on the V2 endpoints (https://www.nuget.org/api/v2)

anangaur added Breaking Change Announcement labels 26 days ago

anangaur locked and limited conversation to collaborators 26 days ago

https://github.com/NuGet/Announcements/issues/37

NuGet server-side API

NuGet talks to a repository

Can be on disk/network share or remote over HTTP(S)

HTTP(S) API’s

- V2 – OData based (used by pretty much all NuGet servers out there)
- V3 – JSON based (NuGet.org, TeamCity, MyGet, Azure DevOps, GitHub repos)

V3 Protocol

JSON based

A “resource provider” of various endpoints per purpose

- **Catalog** (NuGet.org only) – append-only event log
- **Registrations** – materialization of newest state of a package
- **Flat container** – .NET Core package restore (and VS autocompletion)
- Report abuse URL template
- Statistics
- …

https://api.nuget.org/v3/index.json (code in https://github.com/NuGet/NuGet.Services.Metadata)

How does NuGet.org work?

User uploads to NuGet.org

- Data added to database
- Data added to catalog (append-only data stream)
- Various jobs can run over catalog using a cursor
  - *Registrations (last state of a package/version), reference catalog entry*
  - *Flatcontainer (fast restores)*
  - *Search index (search, autocomplete, NuGet Gallery search)*
  - …

Catalog seems interesting!

- Append-only stream of mutations on NuGet.org
  - Updates (add/update) and Deletes
- Chronological
  - Can continue where left off (uses a timestamp cursor)
  - Can restore NuGet.org to a given point in time
- Structure
  - Root https://api.nuget.org/v3/catalog0/index.json
  - + Page https://api.nuget.org/v3/catalog0/page0.json
    - + Leaf https://api.nuget.org/v3/catalog0/data/2015.02.01.06.22.45/adam.jsgenerator.1.1.0.json

NuGet.org catalog

demo

“Find this type on NuGet.org”

- Refactor from using OData to using V3?
  - Mostly done, one thing missing: download counts (using search now)
  - https://github.com/NuGet/NuGetGallery/issues/3532
- Build a new version?
  - Welcome to this talk ☺

Building a new version

What do we need?

- Watch the NuGet.org catalog for package changes
- For every package change
    - Scan all assemblies
    - Store relation between package id+version and namespace+type
- API compatible with all ReSharper and Rider versions

What do we need?

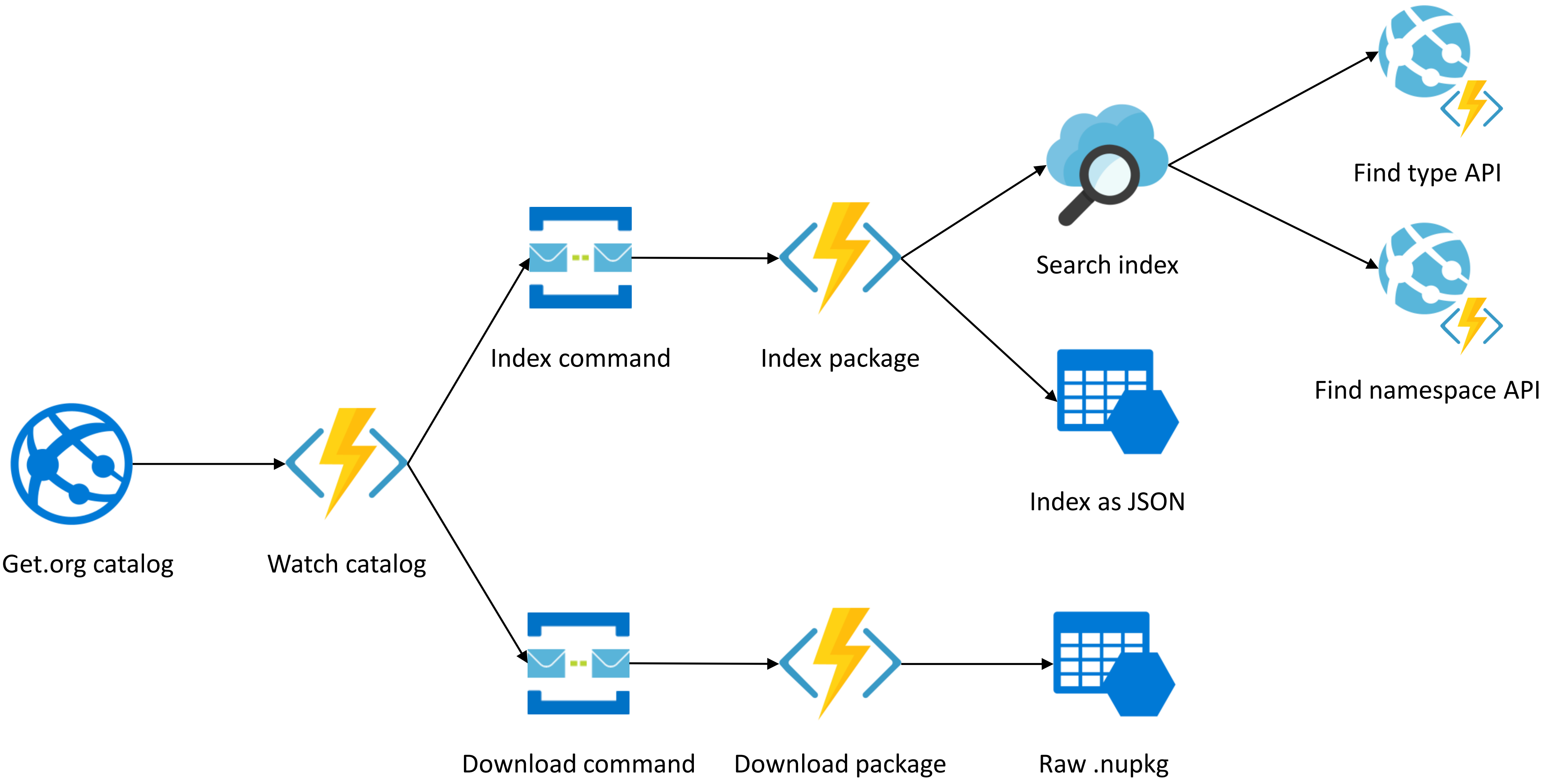
Watch the NuGet.org catalog for package changes periodic check

For every package change based on a queue

- Scan all assemblies
- Store relation between package id+version and namespace+type

API compatible with all ReSharper and Rider versions always up, flexible scale

Sounds like functions!



Collecting from catalog

demo

Functions best practices

@PaulDJohnston https://medium.com/@PaulDJohnston/serverless-best-practices-b3c97d551535

Each function should do only one thing

- Easier error handling & scaling

Learn to use messages and queues

- Asynchronous means of communicating, helps scale and avoid direct coupling

…

Bindings

	Trigger	Input	Output
Timer	✓		
HTTP	✓		✓
Blob	✓	✓	✓
Queue	✓		✓
Table		✓	✓
Service Bus	✓		✓
EventHub	✓		✓
EventGrid	✓		
CosmosDB	✓	✓	✓
IoT Hub	✓		
SendGrid, Twilio			✓
…			✓

Help a function do only one thing

- Trigger, provide input/output
- Function code bridges those

Build your own!*

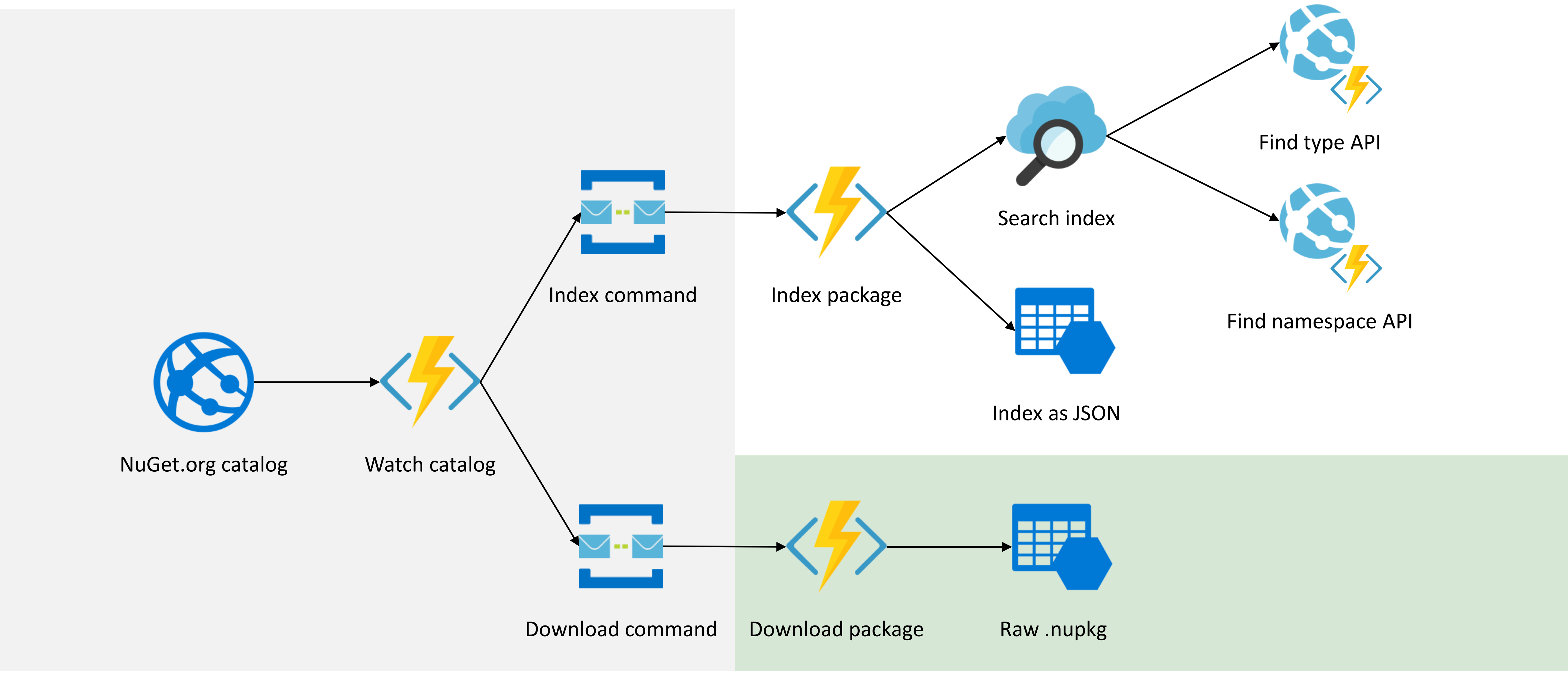
- SQL Server binding
- Dropbox binding
- …
- NuGet Catalog

*Custom *triggers* not officially supported (yet?)

Creating a trigger binding

demo

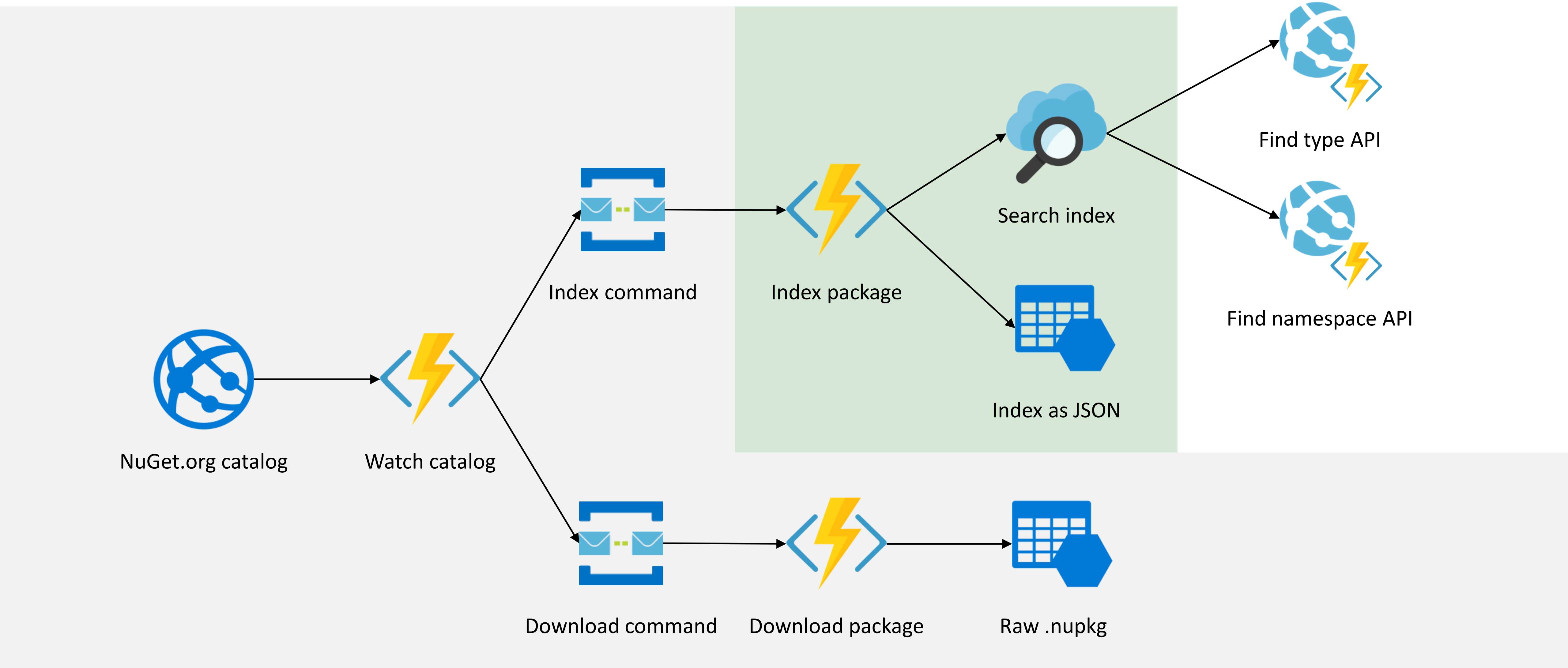
We’re making progress!



Downloading packages

demo

Next up: indexing

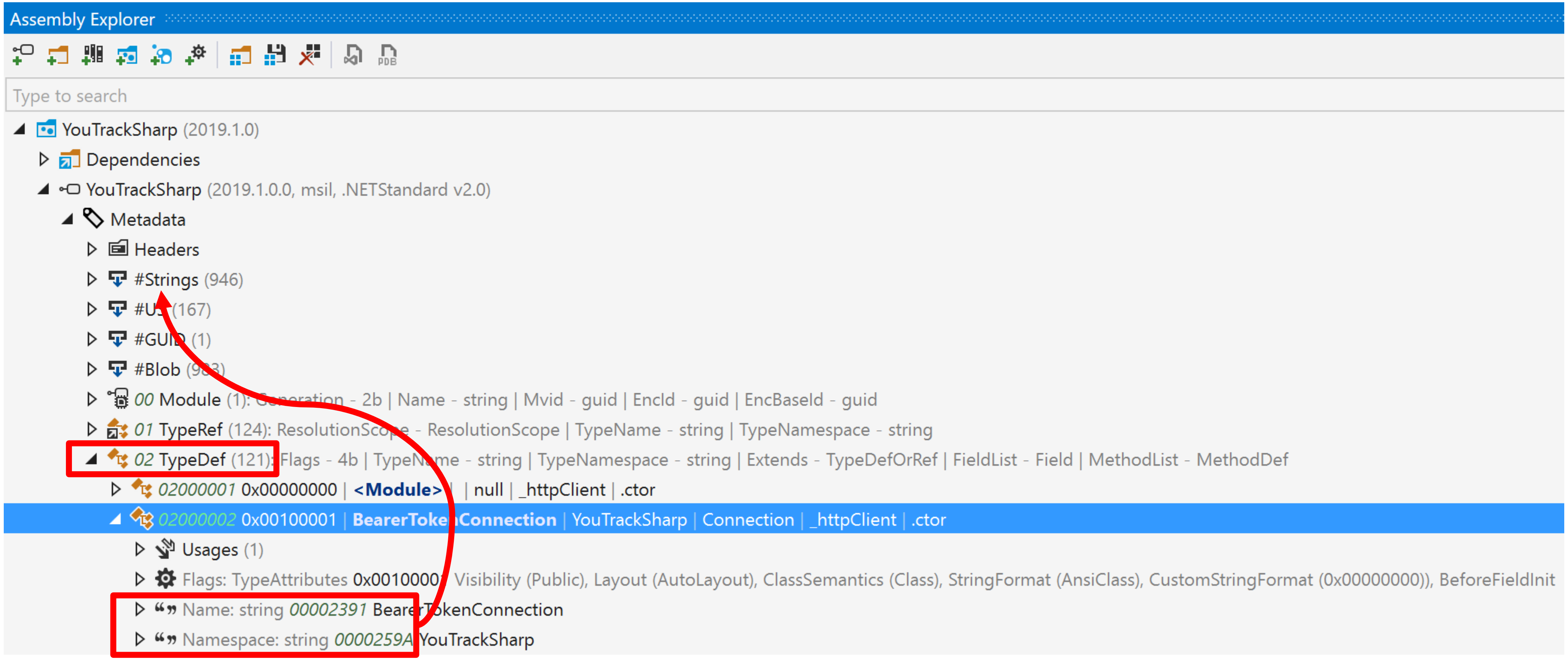


Indexing

- Opening up the .nupkg and reflecting on assemblies
  - `System.Reflection.Metadata`
  - Does not load the assembly being reflected into application process
  - Provides access to Portable Executable (PE) metadata in assembly
- Store relation between package id+version and namespace+type
  - Azure Search? A database? Redis? Other?

System.Reflection.Metadata

Free decompiler
www.jetbrains.com/dotpeek



Assembly Explorer

Type to search

```
◢ YouTrackSharp (2019.1.0)
  ▷ Dependencies
  ◢ YouTrackSharp (2019.1.0.0, msil, .NETStandard v2.0)
    ◢ Metadata
      ▷ Headers
      ▷ #Strings (946)
      ▷ #US (167)
      ▷ #GUID (1)
      ▷ #Blob (983)
      ▷ 00 Module (1): Generation - 2b | Name - string | Mvid - guid | EncId - guid | EncBaseId - guid
      ▷ 01 TypeRef (124): ResolutionScope - ResolutionScope | TypeName - string | TypeNamespace - string
      ◢ 02 TypeDef (121): Flags - 4b | TypeName - string | TypeNamespace - string | Extends - TypeDefOrRef | FieldList - Field | MethodList - MethodDef
        ▷ 02000001 0x00000000 | <Module> |  | null | _httpClient | .ctor
        ◢ 02000002 0x00100001 | BearerTokenConnection | YouTrackSharp | Connection | _httpClient | .ctor
          ▷ Usages (1)
          ▷ Flags: TypeAttributes 0x00100001 Visibility (Public), Layout (AutoLayout), ClassSemantics (Class), StringFormat (AnsiClass), CustomStringFormat (0x00000000)), BeforeFieldInit
          ▷ “” Name: string 00002391 BearerTokenConnection
          ▷ “” Namespace: string 0000259A YouTrackSharp
```

System.Reflection.Metadata

```
using (var portableExecutableReader = new PEReader(assemblySeekableStream))
{
    var metadataReader = portableExecutableReader.GetMetadataReader();
    foreach (var typeDefinition in metadataReader.TypeDefinitions.Select(metadataReader
        .GetTypeDefinition))
    {
        if (!typeDefinition.Attributes.HasFlag(TypeAttributes.Public)) continue;

        var typeNamespace = metadataReader.GetString(typeDefinition.Namespace);
        var typeName = metadataReader.GetString(typeDefinition.Name);

        if (typeName.StartsWith("<") || typeName.StartsWith("__Static") ||
            typeName.Contains("c__DisplayClass")) continue;

        typeNames.Add($"{typeNamespace}.{typeName}");
    }
}
```

Azure Search

“Search-as-a-Service”

Define an index that will hold documents consisting of fields

- Fields can be searchable, facetable, filterable, sortable, retrievable

Structure can’t be changed easily

- Think what we want to search, and what we want to display

Indexing packages

demo

“Do one thing well”

Our function shouldn’t care about creating a search index.

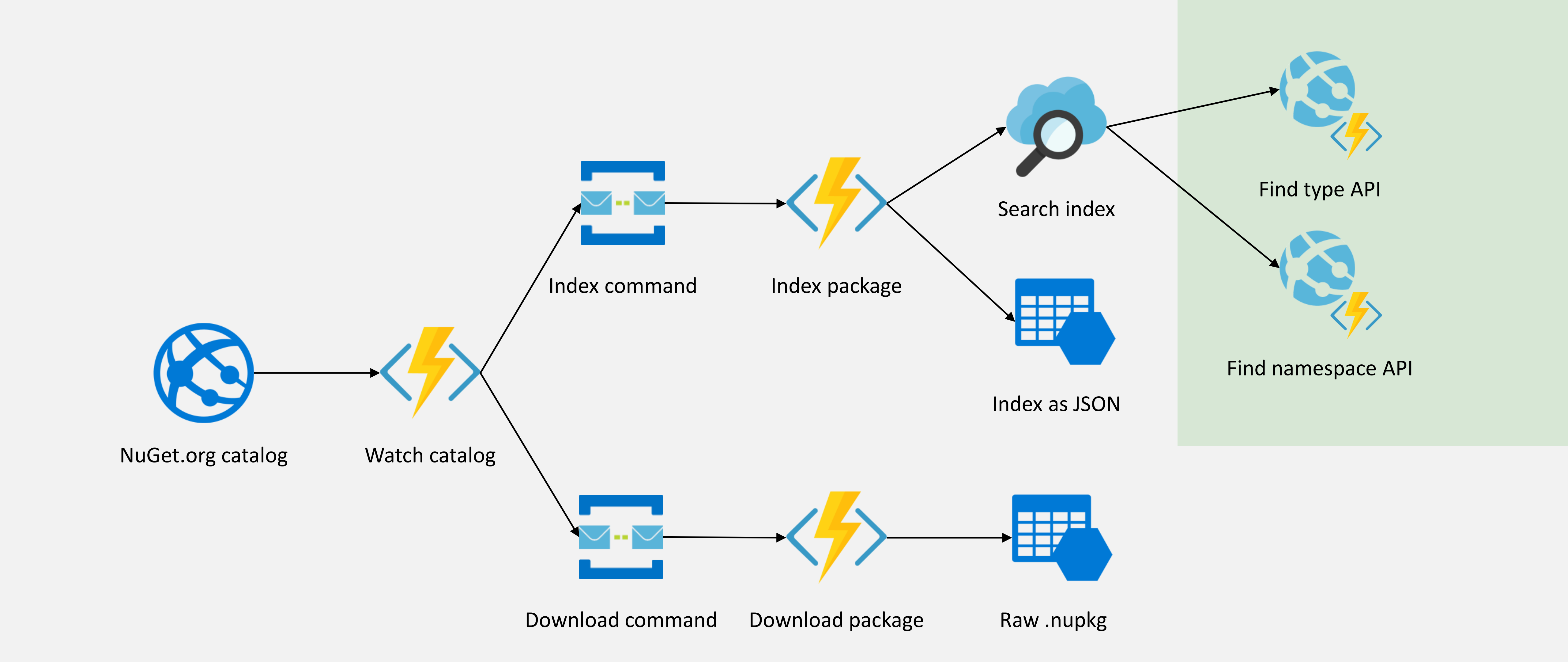
Better: return index operations, have something else handle those

Custom output binding?

Indexing packages (better version)

demo

Almost there...



HTTP trigger binding

```
[HttpTrigger(AuthorizationLevel.Anonymous,
    "get", Route = "v1/find-type")] HttpRequest request
```

Options for trigger

- Authentication (anonymous, a function/host key, a user token)
- HTTP method
- What the route looks like

Making search work with ReSharper and Rider

demo

One issue left...

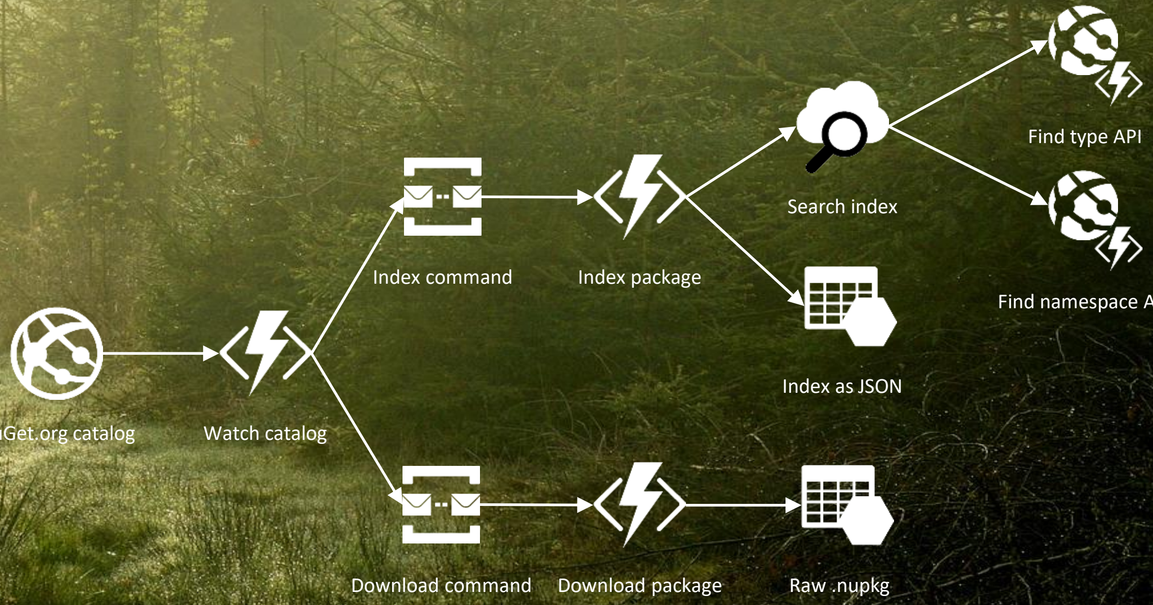
Download counts - used for sorting and scoring search results

- Change continuously on NuGet
- Not part of V3 catalog
- Could use search but that's N(packages) queries
- https://github.com/NuGet/NuGetGallery/issues/3532

If that data existed, how to update search?

- Merge data! new PackageDocumentDownloads(key, downloadcount)

We're done!


Find type API
Search index
Index command
Index package
Find namespace API
Index as JSON
NuGet.org catalog
Watch catalog
Download command
Download package
Raw .nupkg

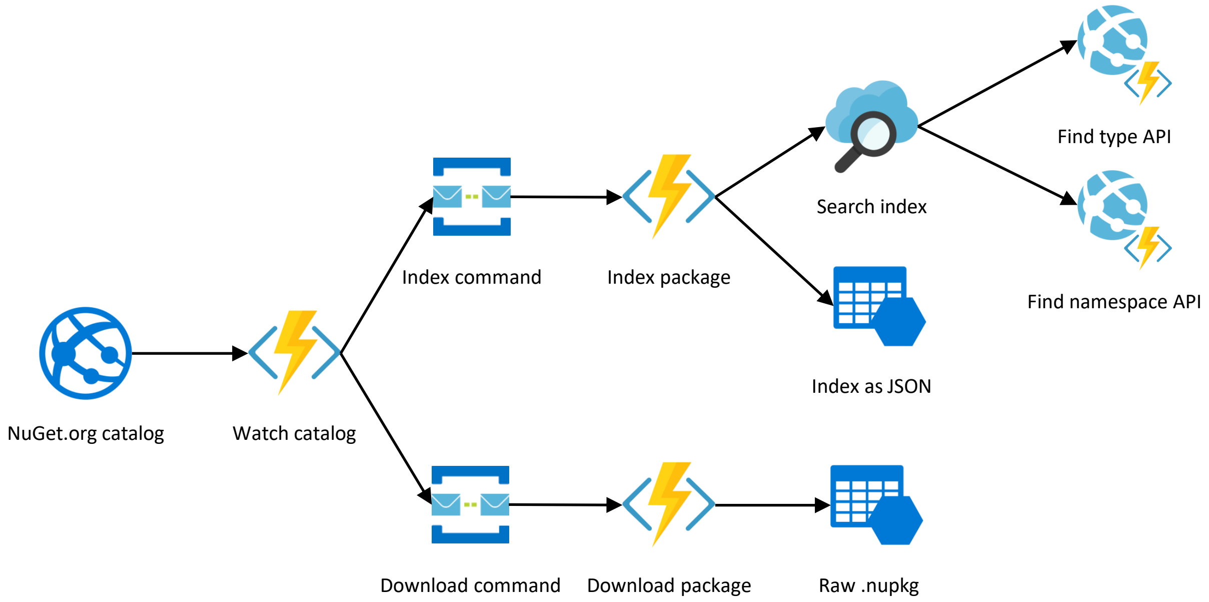
We’re done!

Functions

- Collect changes from NuGet catalog
- Download binaries
- Index binaries using PE Header
- Make search index available in API

Trigger, input and output bindings

- Each function should do only one thing



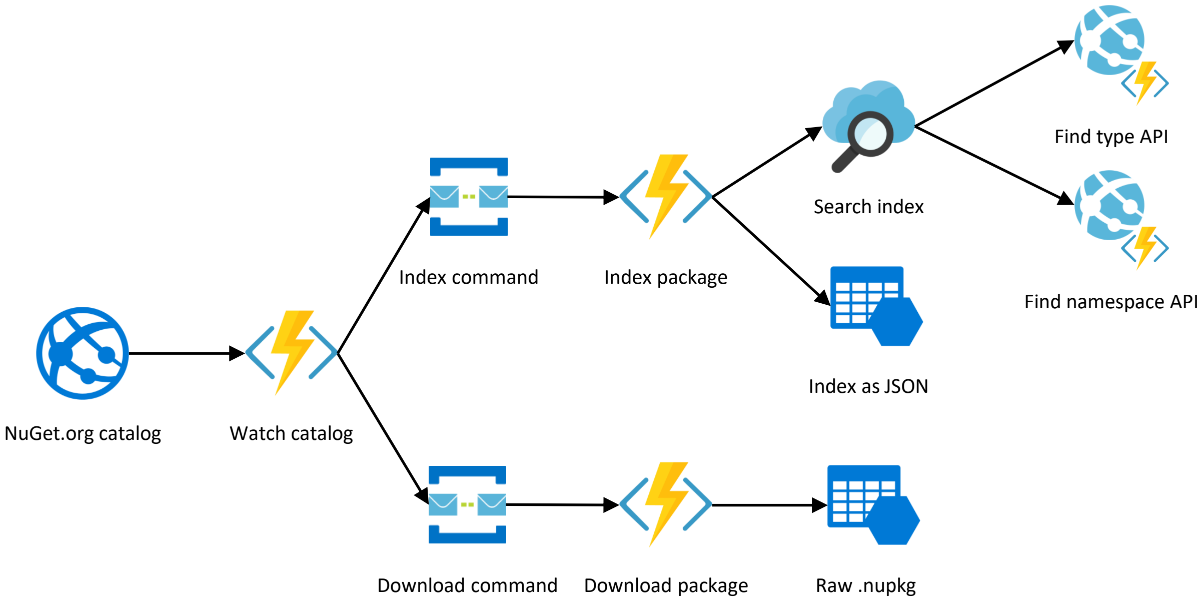
We’re done!

All our functions can scale (and fail) independently

- Full index in May 2019 took ~12h on 2 B1 instances
- ~ 1.7mio packages (NuGet.org homepage says)
- ~ 2.1mio packages (the catalog says ☺)

- ~ 8 400 catalog pages
  - with ~ 4 200 000 catalog leaves
    (hint: repo signing)

- January 2020: ~ 2.6 mio packages / 3.5 TB



NuGet.org catalog

Watch catalog

Index command

Index package

Search index

Find type API

Find namespace API

Index as JSON

Download command

Download package

Raw .nupkg

Closing thoughts…

- Would deploy in separate function apps for cost
  - Trigger binding collects all the time so needs dedicated capacity (and thus, cost)
  - Others can scale within bounds/consumption (think of $$$)
- Would deploy in separate function apps for failure boundaries
  - Trigger, indexing, downloading should not affect health of API
- Are bindings portable…?
  - Avoid them if (framework) lock-in matters to you
  - They are nice in terms of programming model…

Thank you!

https://blog.maartenballiauw.be

@maartenballiauw