Jsonnet: A data templating language from Google

Abhishek Tiwari

Citation: *A. Tiwari*, ”Jsonnet: A data templating language from Google”, Abhishek Tiwari, 2014. doi:10.59350/wg12d-d1479

Published on: October 12, 2014

Jsonnet is a new domain specific configuration language from Google which allows you to define data templates. These data templates are transformed into JSON objects using Jsonnet library or command line tool. As a language, Jsonnet is extension of JSON - a valid JSON object is always valid Jsonnet template.

In terms of use cases, initial focus seems to be on configuration management, package definition, application configuration, etc. It can be also used as template to describe cloud stack, think AWS CloudFormation but more modular and reusable. I have feeling that sooner or later Jsonnet will be used to define, create and manage Google cloud stacks. That said there is nothing which stops you to manage your CloudFormation templates using Jsonnet.

Simplified for humans

Lets see our first hello world Jsonnet template.

```
{
    // Jsonnet example
    person1: {
        name: "Alice",
        welcome: "Hello " + self.name + "!",
    },
    person2: self.person1 { name: "Bob" },
}
```

As you can immediately recognise three subtle differences between JSON and Jsonnet -

1. Jsonnet adds some simple improvements over JSON like commenting which makes it more human readable,
2. Object fields (the strings to the left of the colon) don't have quotes,
3. Referencing - one part of the structure can refer to another part using `self` and **`super`** keywords

Basically Jsonnet is designed to written by human. Let see what is output from above template when we run through the Jsonnet to JSON converter.

```
{
   "person1": {
      "name": "Alice",
      "welcome": "Hello Alice!"
   },
   "person2": {
      "name": "Bob",
      "welcome": "Hello Bob!"
   }
}
```

Data operations

Jsonnet offers more than just above mentioned three features. Lets see another example demonstrating data operations capabilities of Jsonnet. Following snippet combines arithmetic, conditionals, array comprehension

```
{
    foo: [1, 2, 3],
    bar: [x * x for x in self.foo if x >= 2],
    baz: { ["field" + x]: x for x in self.foo },
    obj: { ["foo" + "bar"]: 3 },
}
```

and outputs below JSON object.

```
{
   "foo": [ 1, 2, 3 ],
   "bar": [ 4, 9 ],
   "baz": {
      "field1": 1,
      "field2": 2,
      "field3": 3
   },
   "obj": { "foobar": 3 }
}
```

As you can see, array comprehension in Jsonnet is quite similar to Python array comprehension.

Local Variables

You can define a local variable, a private field using `local` keyword. Unlike fields they use `=` not `:` to attach the value. Local variables can referenced anywhere in the parent object's scope. Local variables will not show up in JSON output but fields they are assigned to will.

```
// outside the object, ends with ";"
local utils = import "myimport.jsonnet";
{
   // inside the object, ends with ","
   local my_var = "This variable is private",
   my_use: my_var

}
```

Modularity

If you have ever used AWS CloudFormation templates for deploying a large stack, it is quite unmanageable due to sheer size of JSON templates. Jsonnet offers construct like imports and functions to avoid complexity of large JSON objects and promote reusability.

Following example illustrates how Jsonnet import works. As you can see `bar_menu.jsonnet` imports `martinis.jsonnet` then concatenation operator + is used to override imported objects from file `martinis.jsonnet`.

```
// martinis.jsonnet
{
    "Vodka Martini": {
        ingredients: [
            { kind: "Vodka", qty: 2 },
            { kind: "Dry White Vermouth", qty: 1 },
        ],
        garnish: "Olive",
        served: "Straight Up",
    },
    Cosmopolitan: {
        ingredients: [
            { kind: "Vodka", qty: 2 },
            { kind: "Triple Sec", qty: 0.5 },
            { kind: "Cranberry Juice", qty: 0.75 },
            { kind: "Lime Juice", qty: 0.5 },
        ],
        garnish: "Orange Peel",
        served: "Straight Up",
    },
}

// bar_menu.jsonnet
{
    cocktails: import "martinis.jsonnet" + {
        Manhattan: {
            ingredients: [
                { kind: "Rye", qty: 2.5 },
                { kind: "Sweet Red Vermouth", qty: 1 },
                { kind: "Angostura", qty: "dash" },
            ],
            garnish: "Maraschino Cherry",
            served: "Straight Up",
        },
        Cosmopolitan: {
            ingredients: [
                { kind: "Vodka", qty: 1.5 },
                { kind: "Cointreau", qty: 1 },
                { kind: "Cranberry Juice", qty: 2 },
                { kind: "Lime Juice", qty: 1 },
            ],
            garnish: "Lime Wheel",
            served: "Straight Up",
        },
    }
}
```

Output JSON object from above Jsonnet template:

```
{
   "cocktails": {
      "Cosmopolitan": {
         "garnish": "Lime Wheel",
         "ingredients": [
            {
               "kind": "Vodka",
               "qty": 1.5
            },
            {
               "kind": "Cointreau",
               "qty": 1
            },
            {
               "kind": "Cranberry Juice",
               "qty": 2
            },
            {
               "kind": "Lime Juice",
               "qty": 1
            }
         ],
         "served": "Straight Up"
      },
      "Manhattan": {
         "garnish": "Maraschino Cherry",
         "ingredients": [
            {
               "kind": "Rye",
               "qty": 2.5
            },
            {
               "kind": "Sweet Red Vermouth",
               "qty": 1
            },
            {
               "kind": "Angostura",
               "qty": "dash"
            }
         ],
         "served": "Straight Up"
      },
      "Vodka Martini": {
         "garnish": "Olive",
         "ingredients": [
            {
               "kind": "Vodka",
               "qty": 2
            },
            {
               "kind": "Dry White Vermouth",
               "qty": 1
            }
         ],
         "served": "Straight Up"
      }
   }
}
```

Now let see an example of functions in Jsonnet. First we define two functions in a file `util_function.jsonnet`, one private and one public. Private functions are defined using `local` keyword. Note the use of `::` which means this is a hidden field and it will not appear in the JSON output.

```
// util_function.jsonnet
local internal = {
    square(x):: x*x,
};
{
    euclidianDistance(x1, y1, x2, y2)::
        std.sqrt(internal.square(x2-x1) + internal.square(y2-y1)),
}
```

Then we import this utility function file in `myfile.jsonnet` which allows access and call to public function `euclidianDistance` but not private function `square`.

```
// myfile.jsonnet
local utils = import "util_function.jsonnet";
{
    crazy: utils.euclidianDistance(1,2,3,4)
}
```

Conclusion

I can already see applications of Jsonnet beyond the configuration management - API management and digital analytics. Jsonnet seems like a perfect choice to write API wrappers. Similarly, data layer objects used in digital analytics domain can be defined using Jsonnet. Initially Jsonnet to JSON converters will be required but eventually systems will be able to accept Jsonnet directly.