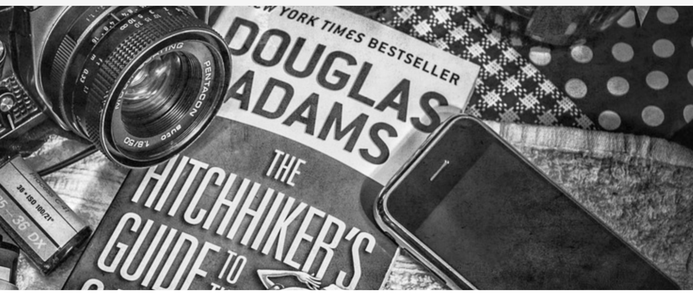


Bambleweeny – it's like WhatsApp, just for machines.

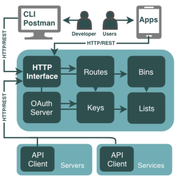
Published on June 8, 2019

Uli Hitzel
Executive Geek

This lightweight HTTP/REST based key-value store & message broker lets you mock APIs and build prototypes in a distributed world. It's fast, easy to use, and well-documented. Here's how you can try this Open Source project yourself!

Kubernetes, Elastic, MongoDB, Kafka, Microservices – we're all dreaming of running planet-scale services, but let's be honest: the scenarios we deal with are often many orders of magnitude smaller. Especially when you build prototypes, conduct Proof of Concepts or run hackathons you need tools that give you quick results and are easy to run.

Last year I've started Bambleweeny, a key-value store & message broker that works over HTTP/REST – in fact, it was originally meant to be just a concept for illustrating the API First Philosophy but it quickly evolved into a real project that now already does its magic in several real-world use cases. What is it, exactly? Technically speaking, it's **Redis with an HTTP interface & OAuth** in a tiny 24 MB **Docker container** that runs (almost) everywhere. You can use it for data ingestion, as a NoSQL database, for mocking API endpoints, caching, webhooks, and coordination across distributed systems – any component can join into the fun and start messaging as long as it's able to perform basic HTTP requests! A quick note on performance: Bambleweeny (or **b9y** for short) easily does 500+ requests per second on an AWS t2.micro instance.

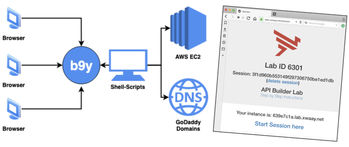


And that's why Bamblweeny is like WhatsApp:

- it **quickly connects** things from all over the place
- it's extremely **easy to use** and it just works
- it's for **fun projects** – you're not supposed to use it for production but might choose to do so, anyway

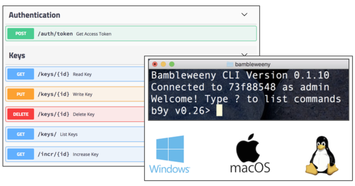
Use Cases

Caleidoscoop is a kiosk system based on this project. A hospitality provider has recently started using Bambleweeny for a prototype where hundreds of systems continuously send data to the 'mothership'. For API Days Singapore I made a hands-on lab around API Builder – but had a challenge: how do I provision resources dynamically and assign to 60 participants in the room? With Bambleweeny it didn't take much longer than a lunch break to make a web-based system that uses message queues for the components to speak with each other.



Everything is API based

I've said it earlier, the API is a first-class citizen in b9y. There is a command line tool for humans, but this one also uses the API to access, create & modify objects. The project is well-documented, however, the Swagger file (which is also available at /swagger) should be sufficient if you'd like to understand how to interact with it, you might even use it to generate client libraries (there is an 'official' one for Python).



Let's mock an API!

Do you have a Docker environment running? Let's take this for a quick spin so you see Bambleweeny in action.

This line gives you b9y running on the default ports:

docker run -d -p 8080:8080 u1ih/bambleweeny

Get the Command-Line Tool for your platform and connect to your b9y instance. Once opened, issue the following commands:

set foo bar

route foo text/html

Copy/paste the output and run this cURL command in a separate terminal window (modify the URL accordingly):

curl http://localhost:8080/routes/7e1545bb-7977-4ee3-8865-bcbb41c19464

The output should be 'bar'. What did we do here? The first step, obviously, created a key named 'foo' with the value 'bar'. According to the API definition, we'd have to do an authenticated request to /keys/foo to retrieve the value. In the second step, we created a **Route** in b9y, which allows anonymous read access to specific keys. There's also **Bins**, which let you push a message to a queue anonymously. The documentation has all the details, let's make this example more interesting and mock a dynamic API endpoint. In the b9y CLI, do the following:

set api '{"message": "!@[message]"}'

set message 'cool stuff!'

route api 'application/json;charset=utf-8'

Notice the odd characters in the first line? It's the dynamic parsing feature, you're basically populating the key 'api' with a skeleton for an API response, which renders the value of 'message' into the response – which is returned along with the Content-Type you specified when creating the route. So a cURL on that route would give you this:

```
$ curl -i http://localhost:8080/routes/3c700b9b-c5b6
-4fa6-8fe7-7261ba4f9d9e
HTTP/1.1 200 OK
B9Y-Routes: parsed
Access-Control-Allow-Origin: *
Content-Type: application/json;charset=utf-8
Content-Length: 26
Date: Sat, 08 Jun 2019 13:57:47 GMT
Server: 0.0.0.0

{"message": "cool stuff!"}$
```