

A TALK ABOUT CONSUL

HELLO!
ROB N ★

Email: robn@fastmailteam.com

Twitter: [@robn](https://twitter.com/robn)

Web: robn.io

robn.io/consul-infracoders-2018

Hi! Thanks for coming! My name is Rob. There's some places you can find me. Do say hi!

These slides are available here, so if I fly past something you were interested in, you'll be able to find it there

HELLO!
FASTMAIL

Email done right, with some heart 

fastmail.com

The very tiny plug for my company, FastMail, who support me fiddling with cool bits of tech and talking to you about it. We care about email and we care about you, so if you'd like your email service with a smile, why not check us out? There's a few FastMailers here, come talk to us later!

CONSUL

So, lets talk about Consul.

CONSUL

- » One instance on every node (host) on your network
- » Working together
- » Tools for managing your services, hosts and networks

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ Consul is a little program that runs on every node on your network

★ They all talk to each other

★ And provide a bunch of tools you can use to run your entire system

CONSUL FEATURES

**SERVICE CATALOG
HEALTH CHECKS
KEY/VALUE STORE
WATCHES
EVENTS
(AND MORE...)**

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Consul has tons of interesting features. I get really excited about it because every time I look there's always something new in there and every time I have a problem, there's often a way to solve it with Consul.

I'm only going to talk about some of these features today.

RAFT

But before we can talk about Consul, we need to spend a couple of minutes talking about Raft, the technology that Consul is built on top of.

RAFT

DISTRIBUTED CONSENSUS ALGORITHM

- » A number of nodes on the network
- » Maintain a "log" (journal) of items
- » Clients submit changes to the log
- » The nodes agree to accept it (or not)
- » Guaranteed to apply the changes on all nodes in the same order

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Raft is a type of algorithm known as a "distributed consensus algorithm".

★ You have some number of nodes on the network. ★ They each maintain a log or journal of items.

★ Clients submit changes to the log, and ★ the nodes talk among themselves and decide to accept the change or not.

★ In a way that ensures that all nodes have the same data set at any given moment.

RAFT

MORE INFO

<https://raft.github.io/>

<https://thesecretlivesofdata.com/raft/>

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Raft is really interesting, but its all math and stuff and knowing how it works isn't necessary for successfully operating Consul, so I'm not going to say any more about it. Here's some places you can find out more.

For our purposes, its enough to say that Consul is built on the assumption that all nodes on the network share a consistent view of the data store.

SERVICE CATALOG

The most important tool Consul provides is the service catalog.

So you have your service, whatever it is. Mine is email. It's made of a number of internal components, usually several instances of each. These talk to each other to do stuff for your users. To talk to each other, they need to find each other! The service catalog is the place to put all of your services and let them find out about each other.

SERVICE CATALOG

- » Register instances of a service (by name)
- » Ask the API where a service is, get an IP:port back
- » Better: ask DNS!

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ So, you register each instance of a service with Consul's service catalog

★ And then you ask the Consul API where some named service is, and it'll tell you. I'll talk about the API a bit later.

★ Consul also has a built-in DNS server, so you can just ask that, making it easy to hook together services that aren't Consul-aware.

SERVICE CATALOG

SERVICES.JSON

```
{  
  "address" : "10.202.2.10",  
  "name" : "hopscotch",  
  "port" : 8084,  
  "tags" : [  
    "beta"  
  ]  
}
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

So here's an example. At FastMail, we have an internal service called "hopscotch". It's a small image proxy so we can embed off-site images in user email without running into browser origin restrictions. Its cool, and entirely unimportant for this talk.

So in Consul's `services.json`, on every node that runs hopscotch, we have a service definition like the above. The name of the service, IP and port, and any tags that let us tell the difference between different flavours of this service. Pretty straightforward?

SERVICE CATALOG

```
$ consul catalog nodes -service=hopscotch
```

Node	ID	Address	DC
betaweb1	8a619b68	10.202.2.10	nyi
qaweb1	2b002a91	10.202.2.11	nyi
web1	66f57633	10.202.2.211	nyi
web2	7d7fcc9a	10.202.2.212	nyi
web3	e47dea91	10.202.2.213	nyi
web4	cf531ebe	10.202.2.214	nyi
web5	b55241eb	10.202.2.215	nyi
web6	7d2d2552	10.202.2.216	nyi

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

We can ask Consul to tell use about all the instances of that service.

SERVICE CATALOG

DNS LOOKUP

```
$ dig +noall +answer hopscotch.service.consul
hopscotch.service.consul. 0 IN A 10.202.2.212
hopscotch.service.consul. 0 IN A 10.202.2.214
hopscotch.service.consul. 0 IN A 10.202.2.10
hopscotch.service.consul. 0 IN A 10.202.2.216
hopscotch.service.consul. 0 IN A 10.202.2.213
hopscotch.service.consul. 0 IN A 10.202.2.11
hopscotch.service.consul. 0 IN A 10.202.2.215
hopscotch.service.consul. 0 IN A 10.202.2.211
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

And even better, we can ask
Consul via DNS, which means we
could put the name
`hopscotch.service.consul`
into some other service, and it
would find its way there without
even knowing Consul is involved.

SERVICE CATALOG

DNS LOOKUP

```
$ dig +noall +answer beta.hopscotch.service.consul  
beta.hopscotch.service.consul. 0      IN  A   10.202.2.10
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Remember how we added tags? We can include them in our lookup to find just flavour we're interested in.

SERVICE CATALOG

consul-template

- » Watch the catalog for changes
- » Rewrite a config file
- » Restart a service

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

We don't always want to use the DNS though. There is a lookup cost, and we can't express everything through DNS, like consistent hashing or ordering. For this, `consul-template` is a useful tool.

★ It's another program that watches the catalog for changes. ★ When a change happens, it rewrites a config file from a template, filling in values it got from the catalog. ★ And it can then take other actions, like restarting the service.

consul-template

```
upstream hopscotch {  
  {{ range service "prod.hopscotch" }} server {{ .Address }}:{{ .Port }};  
  {{ end }}
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

So here's a template for a fragment of nginx config. If you don't know nginx, something it can do is forward web requests on to a pool of backend servers, which it calls "upstreams". So this template builds an upstream block for our hopscotch service. Consul is written in Go, so this uses Go's in-built template language.

consul-template

```
upstream hopscotch {  
  {{ range service "prod.hopscotch" }} server {{ .Address }}:{{ .Port }};  
  {{ end }}
```

```
$ consul-template -template nginx.tpl:nginx.cfg -once
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Here we run `consul-template` in its "generate config and exit" mode.

consul-template

```
upstream hopscotch {  
  {{ range service "prod.hopscotch" }} server {{ .Address }}:{{ .Port }};  
  {{ end }}
```

```
$ consul-template -template nginx.tpl:nginx.cfg -once
```

```
upstream hopscotch {  
  server 10.202.2.211:8084;  
  server 10.202.2.212:8084;  
  server 10.202.2.213:8084;  
  server 10.202.2.214:8084;  
  server 10.202.2.215:8084;  
  server 10.202.2.216:8084;  
}
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

And it produces an nginx
config with all the hopscotch
instances from the catalog.

consul-template

Rewrite config file when catalog changes:

```
$ consul-template -template nginx.tpl:nginx.cfg
```

Rewrite config and run command:

```
$ consul-template  
  -template 'nginx.tpl:nginx.cfg:service nginx reload'
```

Process supervisor!

```
$ consul-template  
  -template nginx.tpl:nginx.cfg -exec 'nginx -c nginx.cfg'
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

There's a bunch of different ways to run it. We can have `consul-template` run forever, watching the catalog for changes, and rewriting the config file when it does.

Here we're having it rewrite the config and then run a command, in this case calling the init system to reload the config.

Or we can use it as a complete process supervisor. In this case when `consul-template` starts, it spawns nginx as a child process and then when the catalog changes it rewrites the config file and sends a SIGHUP to nginx to have it reload. There's lots of options.

HEALTH CHECKS

I keep talking about "when the catalog changes". What makes it change? Health checks are what!

HEALTH CHECKS

- » Checks run on a node
- » Associated with a single service, or the whole node
- » Failing check will mark the service or node "unavailable"
 - » Won't be returned by a catalog query

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

- ★ Each node has its own set of checks
- ★ A check is associated with a single service, or the node as a whole
- ★ A failing check will mark the service or node "unavailable"
- ★ "unavailable" services/nodes won't be returned by a catalog query

HEALTH CHECKS

» Check types:

- » HTTP (Consul asks service regularly)
- » TTL (Service tells Consul regularly)
- » Script (run program, check exit status)
- » Others: TCP, gRPC, Docker

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ There's a few different kinds of checks you can set up.

★ HTTP checks, where Consul makes a HTTP request and looks for a success/fail status

★ TTL checks, where a service calls into Consul regularly, and the check fails if we haven't heard from them in a while

★ Script checks, where Consul runs a program and the return code decides success/fail

★ And others

HEALTH CHECKS

CONSUL.JSON

```
{
  "id": "hopscotch",
  "name": "hopscotch running",
  "service_id": "hopscotch",
  "http": "http://localhost:8084/health",
  "interval": "10s",
  "timeout": "1s"
}
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Here's a simple check for our hopscotch service. This fragment goes in the main Consul config file.

You see the `service_id` there, that's associating the check with the hopscotch service. If this check fails, only that service is made unavailable. Without that, a failing check would make all services on the node unavailable.

It's a HTTP check, that's the URL to hit. If it returns a 200 status, the check passes, otherwise it fails.

And how often to check it, and how long to wait. There's a bunch of other options of course.

HEALTH CHECKS

HTTP CHECK

127.0.0.1 - - [06/Aug/2018:12:50:39 +0000]

"GET /health HTTP/1.1" 200 0 "-" "Consul Health Check"

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

After we add this check, hopscotch starts getting requests to its `/health` endpoint.

HEALTH CHECKS

DNS LOOKUP

```
$ dig +noall +answer hopscotch.service.consul
hopscotch.service.consul. 0 IN A 10.202.2.211
hopscotch.service.consul. 0 IN A 10.202.2.212
hopscotch.service.consul. 0 IN A 10.202.2.213
hopscotch.service.consul. 0 IN A 10.202.2.214
hopscotch.service.consul. 0 IN A 10.202.2.215
hopscotch.service.consul. 0 IN A 10.202.2.216
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

As before, we can see all instances in a catalog query.

HEALTH CHECKS

CHECK FAILING

```
2018-08-06T12:54:40.693711+00:00 web4 consul[79519]:  
agent: Check "hopscotch" HTTP request failed: Get http://localhost:8084/health:  
dial tcp 127.0.0.1:8084: getsockopt: connection refused
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Then one instance fails (in this case, I shut it down by hand). Consul starts reporting the check failure in its log.

HEALTH CHECK

DNS LOOKUP

```
$ dig +noall +answer hopscotch.service.consul
hopscotch.service.consul. 0 IN A 10.202.2.211
hopscotch.service.consul. 0 IN A 10.202.2.212
hopscotch.service.consul. 0 IN A 10.202.2.213
hopscotch.service.consul. 0 IN A 10.202.2.215
hopscotch.service.consul. 0 IN A 10.202.2.216
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Now when we query the catalog,
the failed service is not returned.
That's pretty much all there is to it.
Give your services associated
health checks, and other services
will only ever find ones that are
working right now!

KEY/VALUE STORE

Another major Consul feature is its key/value store.

KEY/VALUE STORE

- » Simple store for service config and other metadata
- » Consistent and available everywhere
- » Not for big data!
- » Not for fast access!

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ It's a simple store for service config and other metadata

★ It's built on top of the Raft log, so its guaranteed to be consistent and available everywhere

★ Before you get too excited, remember that Consul is a service management system, so the KV store is not designed to store big data. It's for small configuration items and service state.

★ And it's not fast! It's designed for consistency, not speed.

KEY/VALUE STORE

```
# consul kv put foo/bar 23  
Success! Data written to: foo/bar
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

It works about how you'd expect. Store some value.

KEY/VALUE STORE

```
# consul kv put foo/bar 23  
Success! Data written to: foo/bar
```

```
# consul kv get foo/bar  
23
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

And pull it back.

KEY/VALUE STORE

```
# consul kv put foo/baz 32  
Success! Data written to: foo/baz  
# consul kv put foo/quux 41  
Success! Data written to: foo/quux
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

The keys are a tree.

KEY/VALUE STORE

```
# consul kv put foo/baz 32
Success! Data written to: foo/baz
# consul kv put foo/quux 41
Success! Data written to: foo/quux

# consul kv get -recurse foo/
foo/bar:23
foo/baz:32
foo/quux:41
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

So you can ask for everything
under some part of the tree.

KEY/VALUE STORE

```
# consul kv export foo/  
[  
  {  
    "key": "foo/bar",  
    "flags": 0,  
    "value": "MjM=",  
  },  
  {  
    "key": "foo/baz",  
    "flags": 0,  
    "value": "MzI=",  
  },  
  {  
    "key": "foo/quux",  
    "flags": 0,  
    "value": "NDE=",  
  }  
]
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

You can dump part of the keyspace as JSON, which is useful for bringing into other tools.

BORING

That's all great, but it's about the minimum of what you'd expect from a KV store. But there's some really nice tools we get from the consistency guarantees Consul provides.

KEY/VALUE STORE CHANGE INDEX

- » Every change bumps the index number
- » Every key records the index number when the key was created, modified and locked
- » Allows efficient synchronisation
 - » You know if the value has changed since you last checked!

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

The store keeps an index number, or change sequence number.

★ Every changes bumps the number

★ Every key records what the index was the last time it was touched

★ This allows easy synchronisation. ★ Your program can come back and say "did this key change since I was last here" or, better, "give me everything that changed since I was last here".

KEY/VALUE STORE

CHECK-AND-SET (CAS)

- » Set this key to X, but only if its index is N
- » Allows higher-level distributed synchronisation primitives (locks, semaphores, shared queues, etc)

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

The KV store offers an atomic "check and set" operation.

★ which means "set this key, but only if it hasn't changed since I last looked at it".

This gives you a way to solve the same concurrent increment problem we have in multithreaded programs, but for distributed systems.

★ And lets you build higher-level synchronisation primitives like distributed locks, shared queues, that sort of thing.

For example, you can implement a lock by trying to get some key, and if it doesn't exist, create it against the current index. If the create works, you know you have the lock. If Consul rejects it because someone else already created the key since you were last there, then you don't have the lock and should try again later.

WATCHES

A couple more quick features, which I've mentioned along the way.

Watches are just a way of saying "block me until some object changes".

WATCHES

- » Wait for something to change
 - » key, keyprefix, services, nodes, service, checks, event
- » Take some action in response

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★★ We can wait on any of Consul's core objects
★ And then take some action.
We looked at `consul-template` earlier; that's exactly what it's doing.

WATCHES

consul watch

```
# consul watch -type=key -key=foo/bar cat
{
  "Key": "foo/bar",
  "CreateIndex": 65791, "ModifyIndex": 65791, "LockIndex": 0,
  "Flags": 0, "Value": "MjM=", "Session": ""
}
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Consul ships with the `consul watch` tool. It lets you create watches from the command line. So here we are making a watch on a single KV key.

WATCHES

`consul watch`

```
# consul kv put foo/bar 99
```

```
Success! Data written to: foo/bar
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Elsewhere, we store a new value on that key.

WATCHES

consul watch

```
# consul watch -type=key -key=foo/bar cat
{
  "Key": "foo/bar",
  "CreateIndex": 65791, "ModifyIndex": 65791, "LockIndex": 0,
  "Flags": 0, "Value": "MjM=", "Session": ""
}
{
  "Key": "foo/bar",
  "CreateIndex": 65791, "ModifyIndex": 69188, "LockIndex": 0,
  "Flags": 0, "Value": "OTk=", "Session": ""
}
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Then we go back to `consul watch`, and see its received the updated key data.

EVENTS

Events are the last feature I want to talk about today.

EVENTS

- » Broadcast a tiny bit of data to Consul nodes
- » Watcher can wait for event, then take action
- » Events are very small, but you can use the KV to coordinate!
- » Example: cache flush

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ Lets your send a message to multiple Consul nodes at once. ★
Then you can have a watcher running on those nodes, and take action when the event arrives.

★ You can actually on fit a tiny amount of data in an event, about 512 bytes. But, you can put data into the KV store before you send the event, and when the event watchers receive it, they can just make a KV query to get the info they need to perform the action.

★ An example we've experimented with a bit is to control cache flushes. Many services cache rarely-changing data in memory. Instead of flushing or rechecking periodically, they set up a Consul watch. When that data does change (some user or administrator action), we send a "flush your cache" event to those nodes. The watches wake up, and those services drop their caches and reload the data.

**ACCESS CONTROLS
SERVICE MESH
NETWORK TOPOLOGY
PREPARED QUERIES
KV TRANSACTIONS
SESSIONS
SNAPSHOTS**

Consul has a ton of other features, and they add something new and interesting in every release. So far, they're holding to the idea of providing a set of building blocks that can be combined in interesting ways.

TOOLS

It can be a bit rough to hand you a box of parts and say "make something", so here's a few tools that come with or work with Consul to get you started.

TOOLS

consul-template

- » Watch for changes in catalog or KV store
- » Rewrite files with new data
- » Run programs, send signals, etc

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

We've already seen consul-template. ★ It watches for changes, ★ rewrites config files, ★ and takes actions, sends signals, other stuff.

TOOLS

consul lock

- » Take a lock
 - » Or block if lock already held
- » Run a program
- » Keep the lock held until the program finishes
- » No more than one copy of the program running across all nodes

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ Takes a lock using a key in the KV store. ★
If it can't get the lock, it sets up a watch and waits until it can.

★ Once it has the lock, runs a program

★ Keeps the lock alive until the program exits (using Consul's "sessions" feature)

★ So you can use it to make sure the program never has more than one running at a time.

TOOLS

consul-replicate

- » Synchronise the KV store to other datacentres
- » Watches `/foo` here
- » Copies to `/remote/foo` there
- » Keep system/application metadata all together in one place

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

I didn't talk about it, but Consul can also operate across multiple datacentres. The catalog is replicated, but the KV is not because there's no way for Consul to know just how much stuff it would be replicating.

★ `consul-replicate` lets you set up cross-DC replication for the KV

★ It watches some key space in the store here

★ And as changes roll in, it stores them in some other key space in the remote DC

★ Lets you keep all your per-node config accessible in one place. For example, we use Consul to record the version of software deployed on each node. By pulling all that information back into our primary datacentre, we can more easily query the KV for info about the entire world

TOOLS

consul exec

- » Run a program on multiple nodes
- » Collect and show the output

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ It's a remote execution tool. Run a program on some set of nodes.

★ Each node stores the output in the KV store, and the calling node sets up a watch to grab it and display it as it comes in.

TOOLS

consul exec

```
# consul exec -node '^wbackup' uptime
wbackup5: 08:44:18 up 59 days, 12:07, 0 users, load average: 0.26, 0.16, 0.23
wbackup5:
==> wbackup5: finished with exit code 0
wbackup3: 08:44:18 up 164 days, 16:53, 0 users, load average: 0.25, 0.84, 0.95
wbackup3:
==> wbackup3: finished with exit code 0
wbackup1: 08:44:18 up 38 days, 10:07, 1 user, load average: 0.79, 0.63, 0.66
wbackup1:
==> wbackup1: finished with exit code 0
wbackup6: 08:44:18 up 25 days, 10:25, 0 users, load average: 1.87, 1.86, 1.85
wbackup6:
==> wbackup6: finished with exit code 0
wbackup2: 08:44:18 up 5 days, 8:59, 0 users, load average: 0.04, 0.04, 0.01
wbackup2:
==> wbackup2: finished with exit code 0
wbackup4: 08:44:18 up 19 days, 11:14, 0 users, load average: 2.06, 1.70, 1.25
wbackup4:
==> wbackup4: finished with exit code 0
6 / 6 node(s) completed / acknowledged
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Here's a basic example. Run the `uptime` command on all of my servers named `wbackup`.

TOOLS

PROMETHEUS

- » Metrics gathering, monitoring and alerting system
- » Reaches out to services on the network and asks them for stats
- » Native support for Consul service discovery

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Prometheus! My other favourite tool at the moment!

★ It's a time-series database with features for collecting and alerting.

★ Works by reaching out to services on the network and asking them what they're up to.

★ And it has native support for Consul service discovery. So once again, it sets up a watch, and when new services appear, it starts to probe them.

TOOLS

PROMETHEUS

```
- job_name: consul
  consul_sd_config:
    - services:
      - node_exporter
      - hopscotch
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

The config is basically nothing,
just the names of the services
you care about.

TOOLS WE GOT ALL KINDS

<https://github.com/josegonzalez/awesome-consul>

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Of course, there's an awesome list of Consul-aware tools and services. Have a look, there's lots of interesting stuff there.

API

And of course, there's an API

API

- » Access to all Consul features
 - » Many not available from command line
- » Simple HTTP+JSON protocol
- » Client libraries for every language

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

★ Complete access to all Consul features, ★ for some things the only way.

★ It's a simple HTTP protocol, with return data in plain JSON

★ There's client libraries for pretty much every language

API SERVICE CATALOG

```
# curl 'http://localhost:8500/v1/catalog/service/hopscotch?tag=prod'
[
  {
    "ID": "66f57633-955f-2136-a5e8-933b1693c4ea",
    "Node": "web1",
    "Address": "10.202.2.211",
    "Datacenter": "nyi",
    "TaggedAddresses": {
      "lan": "10.202.2.211",
      "wan": "10.202.2.211"
    },
    "NodeMeta": {
      "consul-network-segment": ""
    },
    "ServiceID": "hopscotch",
    "ServiceName": "hopscotch",
    "ServiceTags": [
      "prod"
    ],
    "ServiceAddress": "10.202.2.211",
    "ServicePort": 8084,
    "ServiceEnableTagOverride": false,
    "CreateIndex": 75454374,
    "ModifyIndex": 75454374
  }
]
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Here's one example we saw earlier, returning all the available instances of the named services. There's lots of other values we haven't seen yet, but all the interesting bits are there: address, port, tags, and so on.

API KV STORE

```
# curl http://localhost:8500/v1/kv/foo/bar  
[  
  {  
    "LockIndex": 0,  
    "Key": "foo/bar",  
    "Flags": 0,  
    "Value": "OTk=",  
    "CreateIndex": 65791,  
    "ModifyIndex": 69188  
  }  
]
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

KV store fetch is the same
idea.

API CLIENT LIBRARY

```
DB<1> x Consul->kv->get("foo/bar")
0  Consul::API::KV::Response=HASH(0x2f18990)
   'create_index' => 65791
   'flags'        => 0
   'key'          => 'foo/bar'
   'lock_index'   => 0
   'modify_index' => 69188
   'value'        => 99
1  Consul::Meta=HASH(0x3127470)
   'index'        => 69188
   'known_leader' => 1
   'last_contact' => 0
```

Rob N ★ · A talk about Consul · robn.io/consul-infracoders-2018

Or you can use a client library. This is the Perl client library in action from Perl's debugger. I'm using this as the example because I wrote the client library, and I want you to see that Perl is still relevant. But it'll be largely the same in your language of choice.

A TALK ABOUT CONSUL

And that's it. Consul is a whole suite of interesting tools for managing your services, nodes and datacentres. If you're not already using it you should take a look because there might be something in there you can use right now. And if you are using it, look again, because its getting more and more interesting stuff all the time.