



Follow along!

Google Slides: <http://tinyurl.com/lem6l4n> (both letter l's)

Vagrant: <https://github.com/baltimore-puppet-users-group/hierav5>



Going beyond Code

Driving automation with data via Hierac

Dylan Cochran
Systems Engineer
OnyxPoint, Inc



Show me your flowchart and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowchart; it'll be obvious.

– Fred Brooks, The Mythical Man-Month

Smart data structures and dumb code works a lot better than the other way around.

– Eric S. Raymond, The Cathedral and The Bazaar

Bad programmers worry about the code. Good programmers worry about data structures and their relationships.

– Linus Torvalds



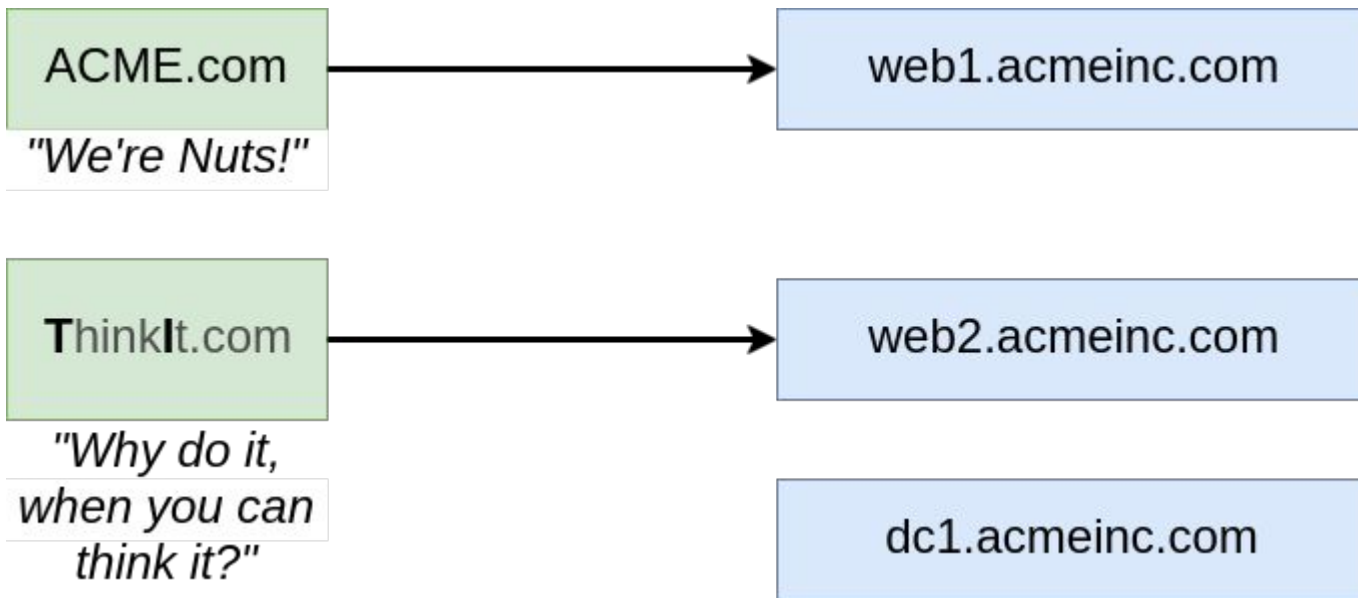
[OVERVIEW]

- Example Organization
- What is Hiera
- Hierarchies
- Environment Layer Data
- Lookup Options
- Delegating control of data
- Module data
- Writing custom backends
- Debugging with 'puppet lookup'



[Example Organization]

ACME Consolidated Mechanical Enterprises, Incorporated





[WHAT IS HIERA - 1]

```
class profiles::ntp() {  
  package { "ntp":  
    ensure => 'installed',  
  }  
  file { ["/etc/ntp.conf":  
    content => "server 192.168.42.5\n",  
  }  
  service { 'ntpd':  
    ensure => 'running',  
    enable => true,  
  }  
}
```



[WHAT IS HIERA - 2]

```
class profiles::ntp() {  
  package { "ntp":  
    ensure => 'installed',  
  }  
  file { "/etc/ntp.conf":  
    content => "server 192.168.42.5\n",  
  }  
  service { 'ntpd':  
    ensure => 'running',  
    enable => true,  
  }  
}
```



[WHAT IS HIERA - 3]

```
class profiles::ntp(  
  $server = "192.168.42.5"  
) {  
  package { "ntp":  
    ensure => 'installed',  
  }  
  file { "/etc/ntp.conf":  
    content => "server ${server}\n",  
  }  
  service { 'ntpd':  
    ensure => 'running',  
    enable => true,  
  }  
}
```




[WHAT IS HIERA - 4]

Puppet automatically converts this portion:

```
class profiles::ntp(  
  $server = "192.168.42.5"  
)
```

Into a puppet function call like this:

```
lookup('profiles::ntp::server', { "default_value" => "192.168.42.5"})
```

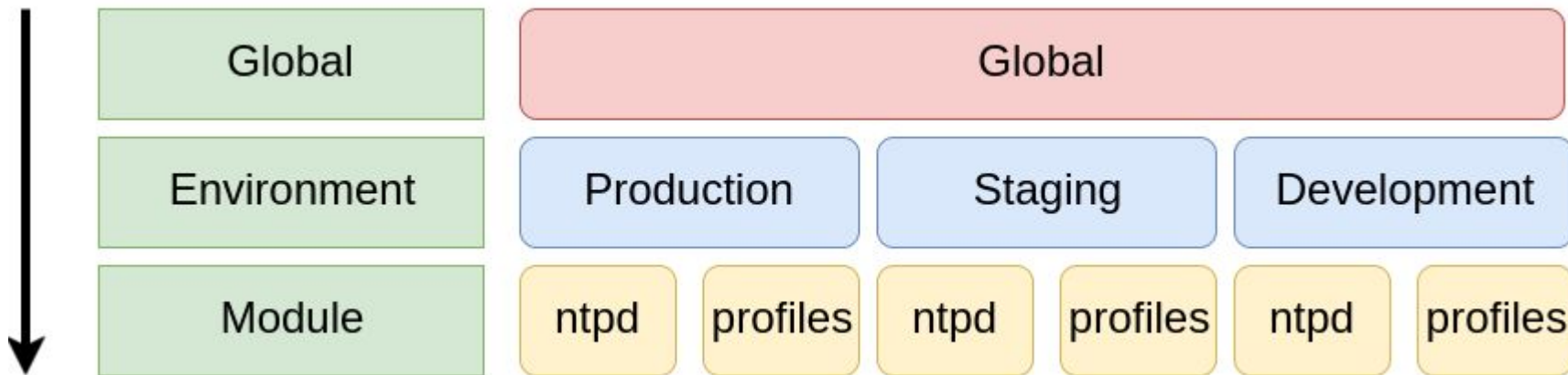
But only when you don't specify the value explicitly!:

```
class { "profiles::ntp":  
  server => '127.0.0.1'  
}
```



[WHAT IS HIERA - 5]

[lookup\(\)](#) calls [Hiera](#) to find a value for 'profiles::ntp::servers'. It looks, in order, in three [layers](#), global, environment, and module. Inside each of those layers are [hierarchies](#) that define where to look for the value.





[HIERARCHIES - 1]

Each one of these layers is configured with a hiera.yaml file

```
---
version: 5                                # Version of hiera
defaults:                                  # used for any level without these settings
  datadir: "data"                          # This path is relative to hiera.yaml
  data_hash: "yaml_data"                  # Use the built-in YAML backend.
hierarchy:                                 # Each hierarchy consists of multiple levels
  - name: "flatfile"                       # Name of the level
    paths:                                  # A list of filenames , relative to datadir
      - "host/{facts.fqdn}.yaml"          # can use variables, facts, and trusted facts
      - "common.yaml"
```



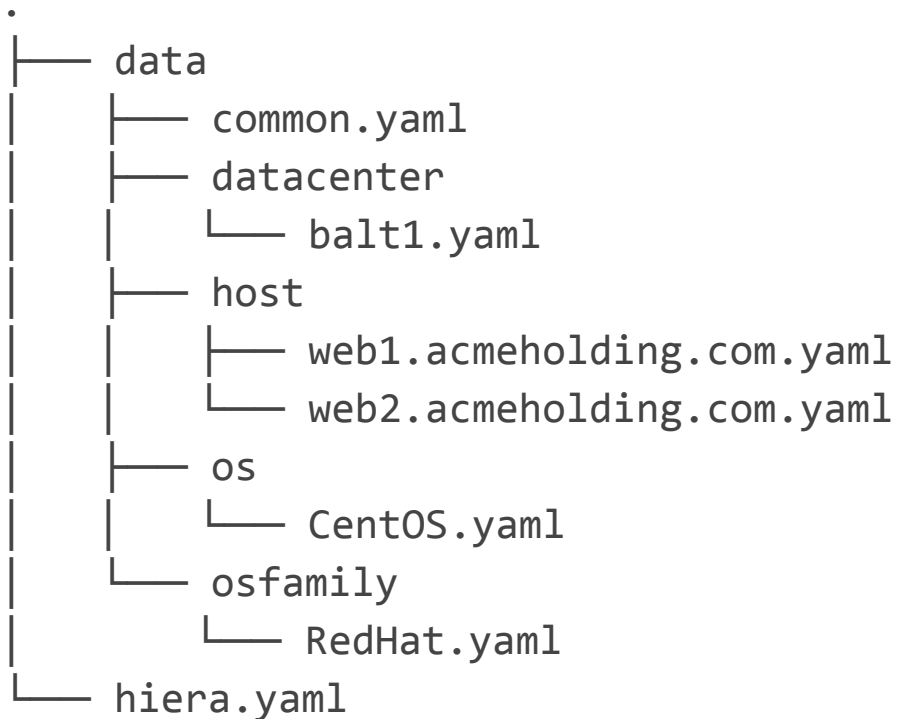
[HIERARCHIES - 2]

hierarchy: # Each hierarchy consists of multiple levels

- name: "OSFamily"
path: "osfamily/{facts.osfamily}.yaml"
- name: "OS Name"
path: "os/{facts.operatingsystem}.yaml"
- name: "Datacenter"
path: "datacenter/{facts.datacenter}.yaml"
- name: "Hostname"
path: "host/{facts.fqdn}.yaml"
- name: "Common"
path: "common.yaml"



[HIERARCHIES - 3]





[ENVIRONMENT DATA - 1]

Let's start doing some real work.

- Both are wordpress sites
- ACME.com is a large team, with an ecommerce component.
- ThinkIt.com has one person.

They've already created a profile to automate the wordpress configuration:



[ENVIRONMENT DATA - 2]

```
class profiles::wordpress(  
  $db_user = undef,  
  $db_password = undef,  
) {  
  include mysql::server  
  include apache  
  include php  
  include apache::mod::php  
  class { "wordpress":  
    install_dir => "/var/www/html",  
    db_user      => $db_user,  
    db_password  => $db_password,  
  }  
}
```



[ENVIRONMENT DATA - 3]

The current best practice is to only use the global layer for site specific overrides. Most data should be in the environment layer.

The global layer are normally located in:

```
/etc/puppetlabs/puppet/hiera.yaml
```

While environment layers are normally in:

```
/etc/puppetlabs/code/environments/<environmentname>/hiera.yaml
```

Global layer is always searched first!



[ENVIRONMENT DATA - 4]

However, in modern puppet installations, you will be using a [control-repo](#) via [r10k](#) or [Code Manager](#).

For a control-repo, just place the hiera.yaml file in the root of control-repo, and the data directory right next to it.

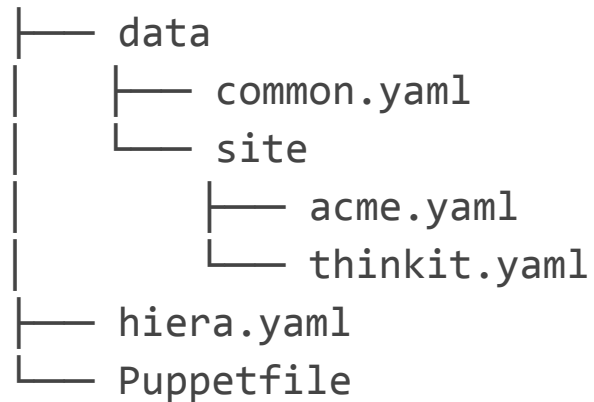
Your data will now be deployed with your puppet code, and branched the same way, for example development, staging, and production.

This means your hiera.yaml configuration is also version controlled at the same time.



[ENVIRONMENT DATA - 5]

/etc/puppetlabs/code/environments/production





[ENVIRONMENT DATA - 6]

version: 5

defaults:

datadir: "data"

data_hash: "yaml_data"

hierarchy:

- name: "flatfile"

Paths:

- "site/{facts.sitename}.yaml"
- "host/{facts.fqdn}.yaml"
- "common.yaml"



[ENVIRONMENT DATA - 7]

Then we'll create two yaml files

data/site/acme.yaml:

```
profiles::wordpress::db_password: "badpa$$w0rd"
```

data/site/thinkit.yaml:

```
profiles::wordpress::db_password: "reallybadpassword"
```



[LOOKUP OPTIONS - 1]

What if we want to do something a little different?

Let's try to classify nodes using only hiera and facts.

First we'll need to create a site.pp file, and give it this content:

manifests/site.pp:

```
$classes = lookup("classes", { "default_value" => []})  
$classes.include
```



[LOOKUP OPTIONS - 2]

Lets first classify every node with profiles::ntp:

```
data/common.yaml:  
  classes:  
    - profiles::ntp
```

And we want to assign profiles::wordpress to acme:

```
data/site/acme.yaml:  
  classes:  
    - profiles::wordpress
```

But then we will lose profiles::ntp, because site/acme.yaml is higher up in the hierarchy.



[LOOKUP OPTIONS - 3]

We can use a special Hieradata key called 'lookup_options'

Every time Hieradata looks up 'classes', it will first attempt to find 'classes' under 'lookup_options', and if it exists it will use those settings.

```
data/common.yaml:
  lookup_options:
    classes:
      merge: unique # This merges arrays from every layer together
  php::extensions:
    merge: hash    # This merges hashes from every layer together
```



[LOOKUP OPTIONS - 4]

Now, when we do a lookup, it will return:

```
classes:
```

- profiles::ntp
- profiles::wordpress



[DELEGATING DATA - 1]

We have separate config files per each site, all within our control repo. Every change is the same as a change in the puppet code.

What if we don't want it to be? What if we want the ACME.com team to be able to configure their database password themselves.

We would have to give them the ability to make changes to the code, or require them to send a pull request every time.



[DELEGATING DATA - 2]

We'll add a new level in the middle of the hierarchy.

hierarchy:

- name: "site data"
path: "site/{facts.sitename}.yaml"
- name: "Delegated to owners"
datadir: "delegated-data"
paths:
 - "{facts.sitename}/common.yaml"
- name: "Common"
path: "common.yaml"



[DELEGATING DATA - 3]

Now we'll create a new git repo (like we make an independent module) for ACME.com. We can then use any permission scheme to delegate control over this git repo.

The only thing this repo would need are yaml files:

```
.
├── common.yaml
└── README.md
```



[DELEGATING DATA - 4]

We'll add a module to our Puppetfile pointing to our code, and tell it to put the code in 'delegated-data' instead of 'modules'

Puppetfile:

```
mod 'puppetlabs-apache', '1.11.0'  
moduledir 'delegated-data'  
mod 'delegated-acme',  
    :git => 'https://github.com/example/hierav5-acme-delegated.git'
```



[DELEGATING DATA - 5]

```
.
├── data
│   ├── common.yaml
│   └── site
│       ├── acme.yaml
│       └── thinkit.yaml
├── delegated-data
│   ├── acme
│   └── common.yaml
├── hiera.yaml
├── manifests
│   └── site.pp
└── Puppetfile
```



[DELEGATING DATA - 6]

One **major** caveat. Every puppet master must be able to see this data. In environments where data **must** be compartmentalized, there is another solution:

hierarchy:

- name: "site data"
path: "site/{facts.sitename}.yaml"
- name: "Compartmented"
paths:
 - "/usr/share/sekrit/{facts.sitename}/common.yaml"
- name: "Common"
path: "common.yaml"



[MODULE DATA - 1]

But what about the module layer?

The module layer acts similar to the environment layer, but with two big restrictions

1. It can only look up keys for that specific module
2. It never touches any other modules data.



[MODULE DATA - 2]

```
class profiles::params {  
  case $::osfamily {  
    'Debian': {  
      $config = "/etc/ntpd.conf"  
      $keys_file = '/etc/ntpd/keys'  
    }  
    'RedHat': {  
      $config = "/etc/ntp.conf"  
      $keys_file = '/etc/ntp/keys'  
    }  
  }  
}
```




[MODULE DATA - 3]

```
class profiles::ntp (  
  $config = $profiles::params::config,  
  $keys_file = $profiles::params::keys_file,  
) inherits profiles::params {  
  
}
```



[MODULE DATA - 4]

Cons

- Data is mixed in code

- Can get very complex over time.

- Encourages not exposing parameters to hiera.

- Confusing for new users



[MODULE DATA - 5]

version: 5

defaults:

datadir: "data"

data_hash: "yaml_data"

hierarchy:

- name: "flatfile"

Paths:

- "osfamily/{facts.osfamily}-{facts.osreleasemajversion}.yaml"
- "common.yaml"



[MODULE DATA - 6]

data/osfamily/RedHat.yaml:

```
profiles::ntp::config: "/etc/ntp.conf"  
profiles::ntp::keys_file: "/etc/ntp/keys"
```

data/osfamily/Debian.yaml:

```
profiles::ntp::config: "/etc/ntpd.conf"  
profiles::ntp::keys_file: "/etc/ntpd/keys"
```



[MODULE DATA - 7]

```
.
├── data
│   ├── common.yaml
│   └── osfamily
│       ├── Debian.yaml
│       └── RedHat.yaml
├── hiera.yaml
└── manifests
    ├── ntp.pp
    ├── params.pp
    └── wordpress.pp
```



[MODULE DATA - 8]

```
class profiles::ntp (  
    $config,  
    $keys_file,  
) {  
  
}
```

Defaults can be in common.yaml, you can also place 'lookup_options' in the module, for sane defaults for hashes and arrays.



[WRITING CUSTOM BACKENDS - 1]

In v5, custom backends are normal puppet functions. They can be written both in Ruby and Puppet.

There are three kinds:

- `data_hash` - Returns every key at once.
- `lookup_key` - Specifies the key you want a value for
- `data_dig` - Specifies keys and subkeys, ex `profiles::ntp::servers.server1.hostname`



[WRITING CUSTOM BACKENDS - 2]

You can specify the function name at any level in the hierarchy, using the type of function it is to specify it:

hierarchy:

- name: "site data"
path: "site/{facts.sitename}.yaml"
- name: "global settings"
data_hash: "profiles::data_hash"
- name: "Common"
path: "common.yaml"



[WRITING CUSTOM BACKENDS - 3]

```
site/profiles/functions/data_hash.pp
function profiles::data_hash(
  Hash $options,
  Puppet::LookupContext $context,
) {
  case $::osfamily {
    'RedHat': {
      {
        "profiles::ntp::config" => "/etc/ntp.conf",
        "profiles::ntp::keys_file" => '/etc/ntp/keys',
      }
    }
  }
}
```



[WRITING CUSTOM BACKENDS - 4]

```
function profiles::lookup_key(  
  Variant[String, Numeric] $key,  
  Hash                                                                $options,  
  Puppet::LookupContext    $context,  
) {  
  if ($key == "profiles::ntp::keys_file") {  
    "/etc/ntp/key2"  
  } else {  
    $context.not_found  
  }  
}
```



[WRITING CUSTOM BACKENDS - 5]

Custom backends can call any function they want, and perform any logic they want.

The only restriction is that custom backends can never create a resource, and if they call lookup, they have to be careful to never call lookup() on the same key.

Puppet will throw an error if you try it, rather than get into an infinite loop.



[DEBUGGING - 1]

Debugging hiera v5 is accomplished with the 'puppet lookup' command.

- 'puppet lookup classes' - will return the classes array for the current node
- 'puppet lookup --explain classes' - will give you debugging output showing which layers and levels it found data
- 'puppet lookup --node web2.acmeinc.com classes' will use puppetdb to look up the facts for web2 and give you the classes array for that node



[DEBUGGING - 2]

```
[root@web2 vagrant]# puppet lookup --explain classes
```

Searching for "classes"

Environment Data Provider (hiera configuration version 5)

Using configuration "/etc/puppetlabs/code/environments/production/hiera.yaml"

Merge strategy unique

Hierarchy entry "site data"

Path "/etc/puppetlabs/code/environments/production/data/site/thinkit.yaml"

Original path: "site/{facts.sitename}.yaml"

Found key: "classes" value: [

"profiles::wordpress"

]

Hierarchy entry "Common"

Path "/etc/puppetlabs/code/environments/production/data/common.yaml"

Original path: "common.yaml"

Found key: "classes" value: [

"profiles::ntp"

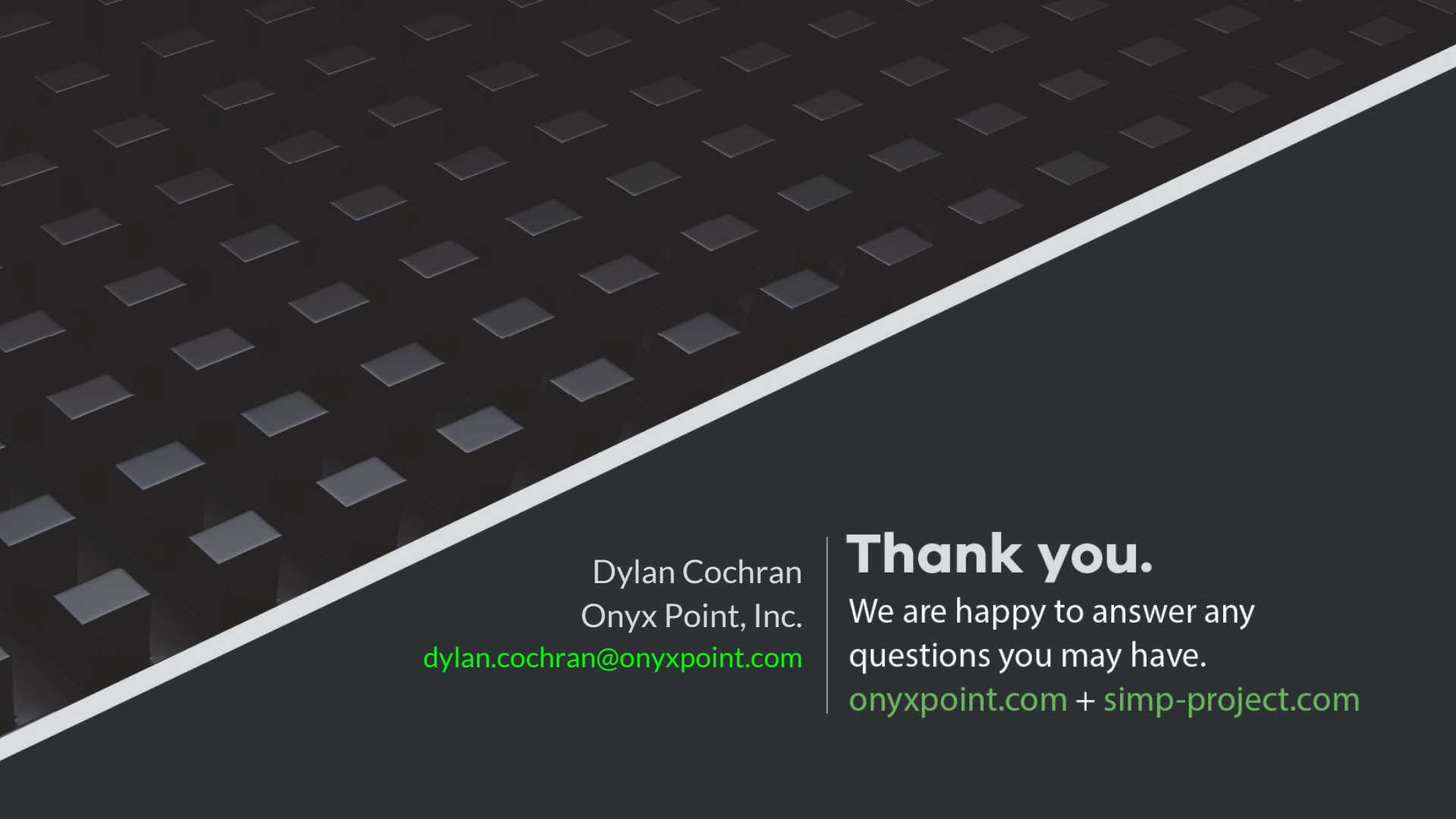
]

Merged result: [

"profiles::wordpress",

"profiles::ntp"

]



Dylan Cochran
Onyx Point, Inc.

dylan.cochran@onyxpoint.com

Thank you.

We are happy to answer any
questions you may have.

onyxpoint.com + simp-project.com