



# Integrating Consul and Puppet

Dylan Cochran  
Systems Engineer  
OnyxPoint, Inc



## [OVERVIEW]

- What is Consul?
- How Consul works
- Setting up a cluster
- Service discovery
- Lock management
- Direct Access in Puppet
- Atomic operations
- Using Consul as a hiera backend



## [WHAT IS CONSUL - 1]

Consul is fundamentally a tool for discovering and configuring services in your infrastructure

- Service Discovery
- Health Checking
- KV Store
- Lock Management
- Distributed Events
- Multi-Datacenter



## [HOW IT WORKS - 1]

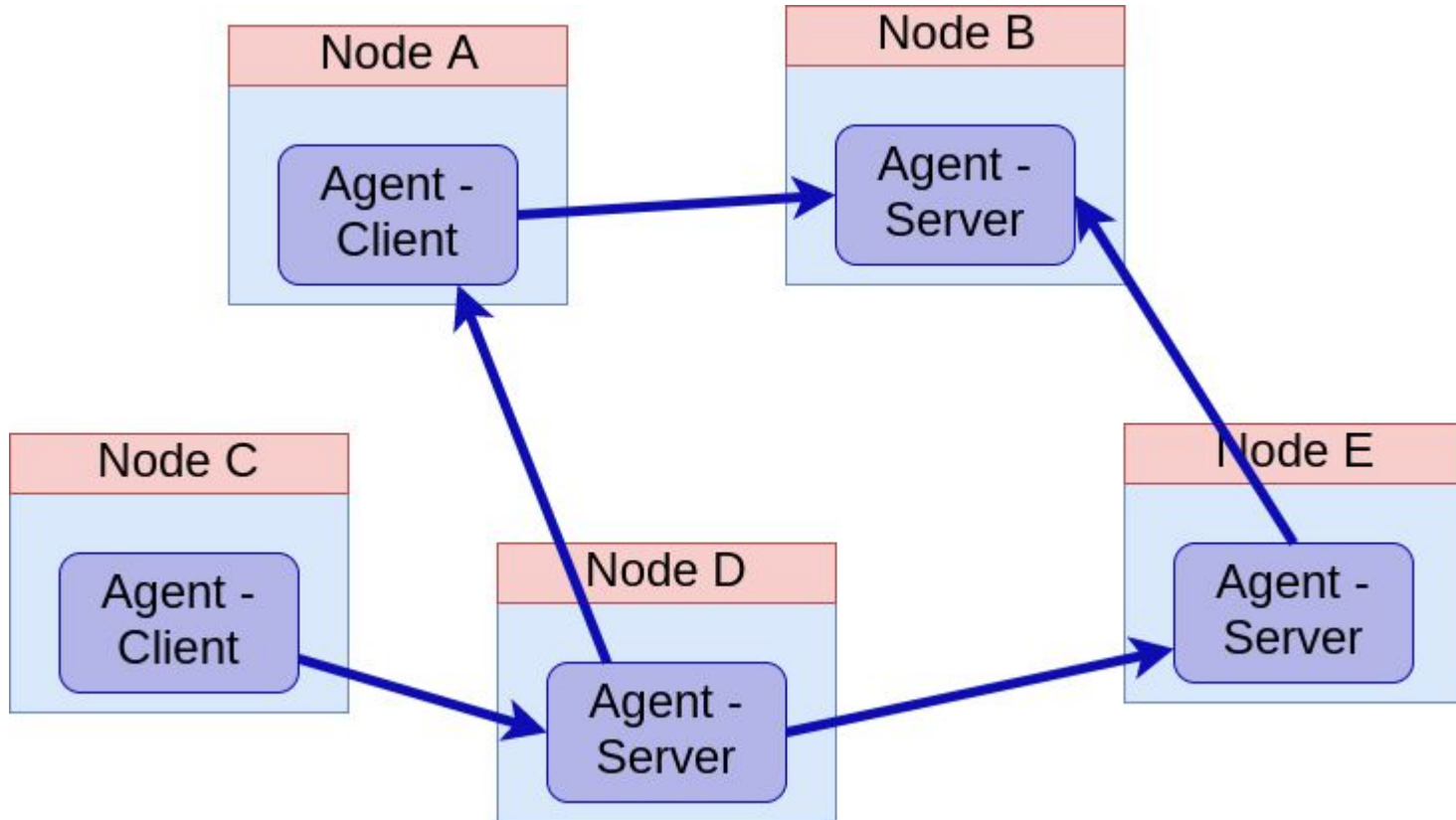
Each node in the cluster runs a consul agent.

All agents attempt to communicate with all other agents using a gossip protocol, to exchange service health information and ip address data.

Every datacenter cluster requires at least one 'server' node.

Servers contain the entire KV store and database, and are considered 'equals'. Normal agents have to connect to a server to access these components.

## [HOW IT WORKS - 2]





## [HOW IT WORKS - 3]

It is recommended to have 3 to 7 server nodes. Consul is strongly consistent, so a majority of server nodes must be available in a datacenter.

1 node = 0 failures

2 nodes = 0 failures

3 nodes = 1 failure

4 nodes = 1 failure

5 nodes = 2 failures

6 nodes = 2 failures

7 nodes = 3 failures



## [SETTING UP A CLUSTER- 1]

We will be using the most well maintained module on the forge:

```
mod 'KyleAnderson-consul'
```

It provides classes to install/configure consul on most platforms, as well as helper types to define consul services and health checks.



## [SETTING UP A CLUSTER- 2]

```
class { "::consul":  
  version => '0.9.2',  
  config_hash => {  
    'node_name' => $::hostname,  
    'advertise_addr' => $::ipaddress,  
    'retry_join' => [ $::ipaddress ],  
    'server' => true,  
    'bootstrap_expect' => 1,  
  },  
}
```





## [SETTING UP A CLUSTER- 3]

Once the first server has been initialized, remove bootstrap-expect. All other nodes (servers and clients) should try to retry-join the first server to get bootstrapped

Afterwards, any joined server is valid to attempt to connect to, as the gossip pool has been started.

You can also enable encryption using the consul keygen command, and adding an 'encrypt' option to the config hash. This will encrypt the gossip pool communication.



## [SETTING UP A CLUSTER- 4]

You can verify the cluster is working by running `consul members` on any agent:

```
dcochran-laptop ~> consul members
```

Node	Address	Status	Type	Build	Protocol	DC
dcochran-laptop	192.168.27.100:8301	alive	client	0.9.2	2	dc1
webserver1	192.168.27.20:8301	alive	client	0.9.2	2	dc1
server1	192.168.27.5:8301	alive	server	0.9.2	2	dc1



## [SERVICE DISCOVERY]

```
consul::service { 'puppet':  
  checks => [  
    {  
      http => 'https://localhost:8140/',  
      interval => '5s',  
    },  
  ],  
  port      => 8140,  
  tags      => ['ca'],  
}
```

We can see the result by querying the dns api on port 8600:

```
dcochran-laptop ~> dig +nocmd +noall +answer -p 8600 @127.0.0.1 ca.puppet.service.consul  
ca.puppet.service.consul. 0 IN  A    192.168.27.20
```



## [LOCK MANAGEMENT - 1]

Consul provides a built-in distributed lock manager, as well as an interface using the `consul lock` command. This can be used to provide exclusive access, as well as 'up to N' levels

```
consul lock -pass-stdin -verbose /test top
```

```
consul lock /production-db-upgrade mysql-db-upgrade
```

```
consul lock -n 4 /puppet-agent puppet agent -t
```



## [LOCK MANAGEMENT - 2]

We can leverage this feature to do rolling restarts of services in puppet:

```
exec { "RestartColdfusion":  
  command => "/usr/local/bin/consul lock coldfusion-restart  
/sbin/service coldfusion restart",  
}
```



## [DIRECT PUPPET ACCESS - 1]

We've created a puppet module, that allows for read/write access to consul directly in puppet code. It provides a set of custom functions that expose a lot of the KV store API.

Libkv Module:

- <https://github.com/simp/pupmod-simp-libkv>



## [DIRECT PUPPET ACCESS - 2]

To start using the functions, you need to specify the url to the consul server:

```
libkv::url: 'consul://localhost:8500/puppet'
```

And adding dependency to your metadata.json:

```
"dependencies": [  
  {  
    "name": "simp/libkv",  
    "version_requirement": ">=0.1.0"  
  }  
],
```



## [DIRECT PUPPET ACCESS - 3]

Classic Exported resources:

```
@@host { $::fqdn:  
  ip => $::ipaddress,  
  tags => [ 'mystuff' ],  
}  
Host <<| tags == 'mystuff' |>>
```





## [DIRECT PUPPET ACCESS - 4]

```
libkv::put("/hosts/${::clientcert}", $::ipaddress)
```

```
$hosts = libkv::list("/hosts")
```

```
$hosts.each |$host, $ip | {  
  host { $host:  
    ip => $ip,  
  }  
}
```



## [ATOMIC OPERATIONS - 1]

What about a system like zookeeper, that requires every server to have a unique id number, between 1-255? We need to be able to allocate these numbers on demand, and ensure that it never gets reallocated

`fqdn_rand()` sounds like a good idea, but it has a tendency to favor lower id numbers, and will never guarantee uniqueness



## [ATOMIC OPERATIONS - 2]

To implement this, we need some way so that when you use 'include zookeeper', it will allocate a unique id to every new server classified, always.

Consul provides a mechanism to do this, which is its 'atomic operations'.

When you use an atomic function, it uses a special object, which contains the index of the last change made, and asserts that the 'put' should only succeed if the index is the same.



## [ATOMIC OPERATIONS - 3]

```
if (libkv::exists("/serverid/${clientcert}")) {  
    $id = libkv::get("/serverid/${clientcert}")  
} else {  
    $oldvalue = libkv::atomic_get("/serverid/current")  
    if ($oldvalue == libkv::empty_value()) {  
        $newvalue = 1  
    } else {  
        $newvalue = $oldvalue["value"] + 1  
    }  
    $result = libkv::atomic_put("/serverid/current", $newvalue, $oldvalue)  
    if ($result == true) {  
        libkv::put("/serverid/${::clientcert}", $newvalue)  
        $id = $newvalue  
    } else {  
        fail("unable to allocate serverid")  
    }  
}
```



## [ATOMIC OPERATIONS - 4]

Remember that in the real world, it can and does fail. If you don't want to fail your catalog compile, you can loop over the result, and re-increment

Because this is such a common process, a `libkv::loop` function is on the roadmap.



## [HIERA - 1]

We can also use consul as a Hiera backend. This will convert hiera keys like 'apache::username' into a consul request to '/puppet/hiera/apache::username'.

It supports all interpolation and layering as other hiera backends

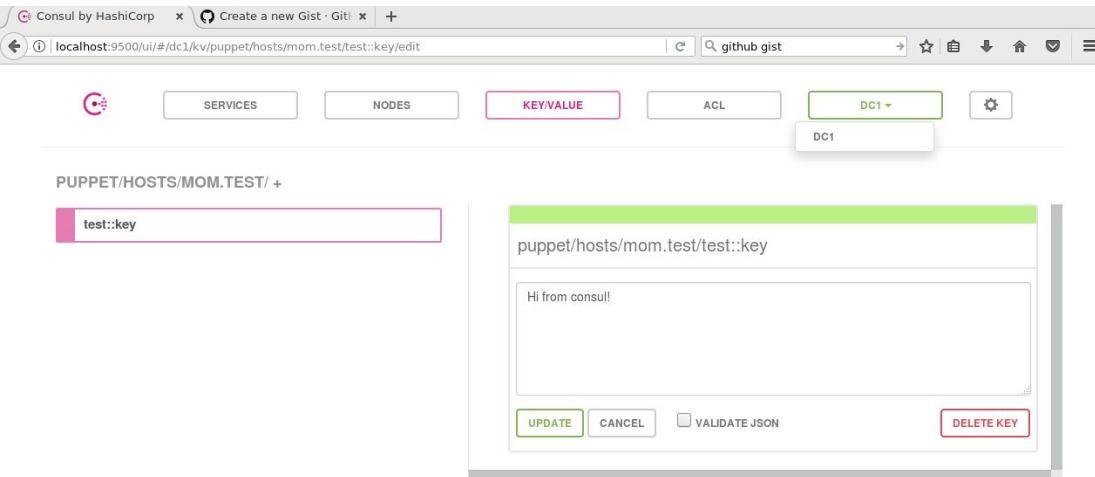


## [HIERA - 2]

```
---
version: 5
defaults:
  datadir: "data"
  data_hash: "yaml_data"
hierarchy:
- name: "Libkv - per-host"
  lookup_key: "libkv::lookup"
  uri: "consul://127.0.0.1:8500/puppet/hiera/hosts/{trusted.certname}"
- name: "Libkv - general"
  lookup_key: "libkv::lookup"
  uri: "consul://127.0.0.1:8500/puppet/hiera/common"
```



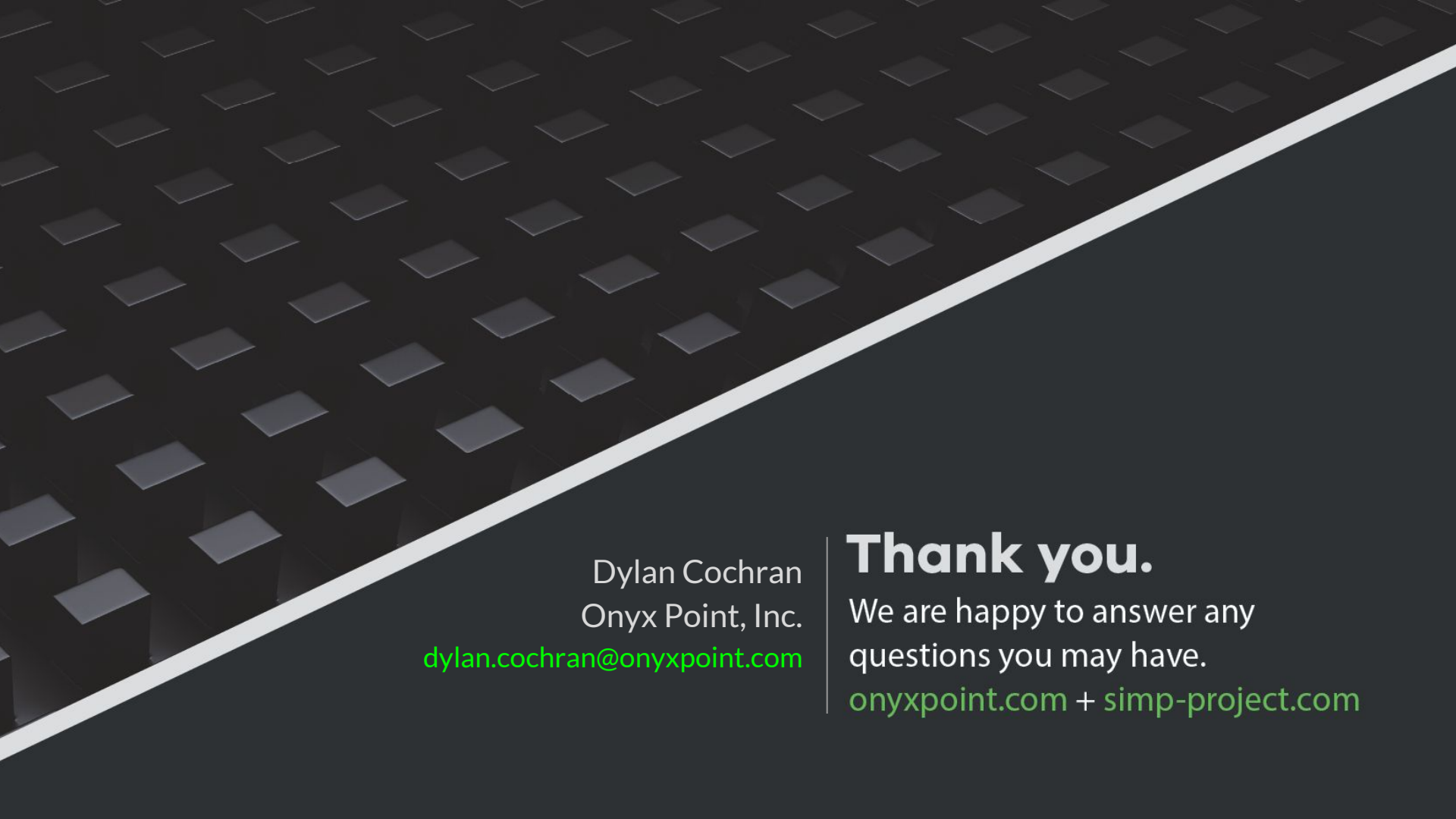
# [HIERA - 3]



```

Searching for "test::key"
Global Data Provider (hiera configuration version 5)
  Using configuration "/etc/puppetlabs/puppet/hiera.yaml"
  No such key: "test::key"
Environment Data Provider (hiera configuration version 5)
  Using configuration "/etc/puppetlabs/code/environments/development/hiera.yaml"
  Hierarchy entry "Libkv - per-host"
    URI "consul://127.0.0.1:8500/puppet/hosts/mom.test"
    Original uri: "consul://127.0.0.1:8500/puppet/hosts/%{trusted.certname}"
    Found key: "test::key" value: "Hi from consul!"
[root@mom vagrant]#
  
```





Dylan Cochran  
Onyx Point, Inc.

[dylan.cochran@onyxpoint.com](mailto:dylan.cochran@onyxpoint.com)

**Thank you.**

We are happy to answer any  
questions you may have.

[onyxpoint.com](https://onyxpoint.com) + [simp-project.com](https://simp-project.com)