Managing Hundreds of MySQL Servers Efficiently

http://bit.ly/puppet-mysql-slides

https://github.com/mozilla-it/puppet-mysql

Brandon Johnson
@cyborgshadow
Sheeri Cabral
@sheeri
Mozilla Database Engineering

The Challenge

- Consistency – no one offs!
- Config files
- Scripts
- Auxiliary files

As little human interaction as possible

In case of disaster,

how long does it take to get another server

built exactly the same?

AWS without EBS...and even with it!

Not just one MySQL

- MySQL
- Percona's patched MySQL
- MariaDB
- Tokutek
- Different versions

Config Files

- Backup restores just work
  - e.g. innodb_log_file_size
- Sane defaults
  - e.g. 75% of RAM to innodb_buffer_pool_size
- As little human knowledge as possible

Scripts

- Backup scripts
- ETL
- Data refreshes
- Monitoring
  - alerting
  - trending

Auxiliary Files/Settings

- sysctl
  - File/proc
- swappiness
- mount options
  - e.g. noatime

Anything else?

What to Configure?

- Files
- OS settings
- Packages
- Dependencies

Dependencies

- Software
  - OS version (RHEL5 vs. RHEL 6)
  - Bugs/features in software (pt-table-checksum 2.1.8)
  - Compatibility (perl-DBD-MySQL-4.0.13)
- Hardware
  - Battery-backed write cache
  - RAM
  - # CPUs

Sometimes more than one of these!

RHEL5

Percona toolkit 2.1.8

perl-DBD-MySQL-4.0.22

Goal:

To be able to spin up a new server with minimal interaction, similar or exact

e.g. might be RHEL6 instead of RHEL5

Software

- Use repos
  - Can make internal ones
  - e.g. Oracle's MySQL
- Software fork and version are dependencies

Fork/Version Determine

- Repos
- Packages
- Service name
    - mysql
    - mysqld
    - mysql.server

$package_type

- mysql56
- maria55
- percona55
- tokutek
- percona51
- mysql [5.0]

Server Packages

```
$packages = $package_type ? {
    "mysql56" => ["MySQL-server-5.6.12"],
    "mariadb55" => ["MariaDB-server"],
    "percona55" => ["Percona-Server-server-55"],
    "percona55" => ["Percona-Server-server-51"],
    "mysql" => ["mysql-server"],
```

Service Name

```
$service_name = $package_type ? {
    "mysql56" => "mysql",
    "mariadb55" => "mysql",
    "percona55" => "mysql",
    "percona51" => "mysql",
    "tokutek" => "mysql.server",
    "mysql" => "mysqld",
}
```

Congratulations!

You have now seen our settings.pp file
aka the “settings” class

Instantiation

- aka “node manifest”

```
node /^dev[12].db.phx1.mozilla.com$/ {
    $mysql_package_type = 'mysql56'
…
```

- Not a class instantiation of settings
- That happens inside other classes
- We will see this later

Client Packages

- In the “client” class, client.pp
- Separated because we may want to instantiate only mysql::client
  - If there is no server
  - e.g. administrative machine
- Client, shared libraries, etc

client.pp

```
# we'll need this regardless, for percona-toolkit
realize(Yumrepo["percona"])
$client_package_name = $mysql::settings::package_type ? {
```

- Note the call to the settings class
- Packages are not yet installed
- Only setting an array
- settings.pp does the same

client.pp

```
# we'll need this regardless, for percona-toolkit

realize(Yumrepo["percona"])

$client_package_name = $mysql::settings::package_type ? {

    "mysql56"   => ["MySQL-client-5.6.12", "MySQL-shared-compat-5.6.12",
"MySQL-shared-5.6.12"],

    "percona55" => ["Percona-Server-client-55", "Percona-Server-shared-
compat"],

    "mariadb55" => ["MariaDB-client","MariaDB-compat","MariaDB-
common"],

…

   default     => "mysql",

 }
```

Where to get packages from

- Define repos
- Gotcha: MariaDB vs. MySQL
- MariaDB pkgs replace MySQL
- ensure => absent

Where to get packages from

```
if $package_type == 'mysql56' {
    yumrepo { 'mariadb':  ensure => absent   }
    realize(Yumrepo['mozilla-mysql'])
}
if $package_type == "mariadb55" {
    realize(Yumrepo["mariadb55"])
}
```

- Percona repo already realized

Requirements for Installation

- We just saw code to set up (“realize”) the repo
- To be safe, require the repo before going further
- This will include Percona's repo
- Repos are dependencies for installation

Requirements for Installation

```
package {   $client_package_name:
        ensure          => present,
```

- This is what installs the package
- $client_package_name is the array set previously

Requirements for Installation

```
package {   $client_package_name:
      ensure          => present,
      require         => $package_type ? {
         "mysql56"   => Yumrepo["mozilla-mysql"],
         "percona55" => Yumrepo["percona"],
         "mariadb55" => Yumrepo["mariadb55"],
         "tokutek"   => undef,
         "percona51" => Yumrepo["percona"],
         default     => undef,          }   }
```

Install Packages Globally

```
package {
        'percona-xtrabackup':
            ensure => present;
        'percona-toolkit':
            ensure => "2.1.8-1",
            require => Package[$client_package_name];
    }
```

That was our client.pp file
aka the “client” class

That was our client.pp file
aka the “client” class

Well, ours has some code about setting a root password if the cluster isn't defined and /root/.my.cnf doesn't exist

The big guns:
server class, server.pp

The big guns:
server class, server.pp
Similar logic to client.pp
but way more stuff

Server Packages

- Like client packages
- More logic/dependencies

Files

- What files/directories on all servers?
  - Global scripts
  - /var/lib/mysql/, /var/log/mysql/, /var/run/mysqld/
  - authorized_keys
  - /etc/security/limits.d file/proc limits

server class is bigger
because it solves more problems

We showed packages

We showed packages

files/directories standard in config mgmt

We showed packages

files/directories standard in config mgmt

biggest issue left is configuration files!

Config files

- Can hard-code standards
- Or have defaults with overrides
- Overrides for things that change
- Sane defaults
    - Can be based on hardware
        - RAM
        - Battery-backed write cache
- Actually defined in server.pp

Config files

- Called in server.pp
- Variables are passed through server.pp

```
file {
    '/etc/my.cnf':
        owner => "mysql",
        group => "mysql",
        content => template("mysql2/my.cnf.erb"),
        require => Package[$mysql2::settings::packages],
        before => Service[$mysql2::settings::service_name];
}
```

Template Config File

- Hard-coded vs. variable-based

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
innodb_file_per_table
<% if server_role == 'slave' %>
read_only=ON
<% else -%>
read_only=OFF
<%end%>
```

Template Config File

- Hard-coded default w/override

```
<% if expire_logs_days != :undef -%>
expire_logs_days=<%= expire_logs_days %>
<% else -%>
expire_logs_days=10
<% end -%>
```

Template Config File

- Variables based on server info

```
log-bin=/var/lib/mysql/<%= scope.lookupvar('::hostname') %>-bin
```

- With override

```
<% if innodb_buffer_pool_size != :undef -%>
innodb_buffer_pool_size=<%= innodb_buffer_pool_size %>
<% else -%>
innodb_buffer_pool_size=<%= (memorysize.split(' ')[0].to_i*1024)/2 %>M
<% end -%>
```

Template Config File

- Array or none

```
<% if replicate_wild_do_table != :undef -%>
  <% replicate_wild_do_table.sort.each do |value| -%>
  replicate_wild_do_table=<%= value %>
  <% end -%>
<% end -%>
```

Instantiation

```
node /^dev[12].db.phx1.mozilla.com$/ {
    $mysql_package_type = 'mysql56'

    class {
        'mysql2::server':
            server_role => $::fqdn ? { /^dev1/ => 'master', default =>
    'slave', },
            cluster                 => 'dev',
            innodb_buffer_pool_size => '8G',
            binlog_format           => 'MIXED',
            expire_logs_days        => '7',
            wait_timeout            => '120',
            key_buffer_size         => '1G',
            swappiness              => '30';
```

Any missed logic?

Easy to clone a server's configuration

Easy to clone a server's configuration

Exactly

Easy to clone a server's configuration

Exactly

Or for a different cluster

Multipurpose Variables

- $server_role
  - read_only in /etc/my.cnf
  - used in setting /etc/motd
- $cluster
  - also used in setting /etc/motd
  - root password along with .my.cnf presence
- /etc/motd
  - MySQL $server_role for $cluster

OS Configuration

- By calling other packages with variables

```
sysctl::value {
   'vm.swappiness':
    value => $swappiness; }
```

- If other classes exist
- If not yet, manually in the instantiation - mounts

Instantiation

```
node /^dev[12].db.phx1.mozilla.com$/ {
    $mysql_package_type = 'mysql56'
    mount { '/':
        ensure  => mounted,
        atboot  => true,
        fstype  => 'ext4',
        before  => Class['mysql2::server'],
        options =>
'errors=remount-ro,noatime,nodiratime,barrier=0',
        device  => '/dev/sda3';
    }
```

Manual mounting is a Hack

- Another 7 lines for **each** node description
- Better than nothing
- Just barely
- Short-term until there is a separate class for it

Grants

- Grant statement has 7 parameters
- GRANT *priv* ON *db*.*tbl* TO *user@host* IDENTIFIED BY *password [WITH GRANT OPTION]*
- Grant class just uses these, “dumb”
- Plus $revoke to revoke the grant

Grants

```
if ($revoke) {
  exec{ "mysql2::grant::${name}":
   command => "mysql -e
  \"REVOKE ${privileges},${grantrevoke}
   ON ${database}.${tables}
    FROM ${username}@'${host}';
      FLUSH PRIVILEGES\"",
  onlyif => "mysql -e
  \"SHOW GRANTS FOR '${username}'@'${host}'\"
    | tr -d \"'\\\`\" | grep -i
   \"GRANT ${privileges} ON ${database}.${tables}
  TO ${username}@${host}${grantopt}\"",
 require     =>
  Service[$mysql2::settings::service_name];
}
```

Grants

```
else {
  exec { "mysql2::grant::${name}":
   command     => "mysql -e \"GRANT $
  {privileges} ON ${database}.${tables} TO
   ${username}@'${host}' IDENTIFIED BY
  '${password}' ${grantopt};
   FLUSH PRIVILEGES\"",
                  unless      =>
[same show grants as the revoke's “onlyif”]
…
                  require     =>
  Service[$mysql2::settings::service_name];
```

Grants class

- Does one thing
- Does it well
- Called in other classes
- Can call in instantiation
- If MySQL introduces new privileges, it is ready

Grants class

- Called in server class
- For global users (e.g. monitoring)
- Instantiate for groups
  - dev machines and user(s)

Remember

- Consistency

- Easily create new instance
  - Exactly the same
  - Similar
  - e.g. stage might have a stage user
  - Copy/paste or copy/paste and edit

Databases class

- Class for ensuring a db exists
- Very simple

```
exec { "create-${name}":
 unless      => "mysql ${name}",
 command     => "mysql -e \"create database ${name}\"",
 path        => ["/bin", "/usr/bin", "/usr/local/bin"],
 environment => ['HOME=/root'],
 require     => Service[$mysql2::settings::service_name],
}
```

Databases class

- Can add a user to it

```
if $username != undef and $password != undef {
mysql2::grant {
    $name:
        username => $username,
        password => $password,
        database => $name;
}
```

- Can also add $host, $grant [privs], other vars

Ensure a Runtime Variable

- Variable class
- Require the service is running

```
define mysql2::variable($value) {
exec { "mysql2::variable::${name}":
 command => "mysql -e \"SET GLOBAL
                              ${name} = ${value}\"",

 unless =>
  "mysql -e \'SHOW VARIABLES LIKE \"${name}\"'
              | grep  ${name}.*${value} | wc -l",
    }
}
```

Example: server.pp

- Global example
- Can use the same syntax in an instantiation

```
if ($slowlogs) {
mysql::variable {
  'slow_query_log': value => 'ON';
  'slow_query_log_file': value =>
 $slowlogs_logfile;
  'long_query_time': value => $long_query_time; }

} else {
  mysql::variable {
  'slow_query_log': value => 'OFF'; }}
```

Runtime Variables

- Alert if runtime vs. config file is different
  - We use Nagios
- pt-config-diff
- False negatives
  - pt-config-diff compares runtime vs. config
  - not runtime vs. default

Scripts, crons, init scripts

- Big source of one-off pain
- Big problem if they disappear
- Big win to have these in version + config control

Scripts, crons, init scripts

- Scripts, crons, inits all similar
- A file copied to a location
- Optional parameters
  - To be set in instantiation

At Mozilla

- Consistent paths
- Different paths
- /etc/cron.d for crons
- /usr/local/bin for scripts
- /etc/init.d for inits
- Can be overridden

Script Paths

- Defaults, can override

  $script_path = '/usr/local/bin',

  $cron_path = '/etc/cron.d',

  $init_path = '/etc/init.d',

- Your script paths may vary, check:
  - ${operatingsystem}-${operatingsystemrelease}
  - e.g. RedHat-6

Script Variables

- Defaults to cron + script:

  $want_script = true,

  $want_cron   = true,

  $want_init   = false,

- At least one should be true
  - Logic in script.pp

If $want_script is true

- Ensure the script is present
- Set the source

```
content => template("${source_prefix}/${name}");
$source_prefix = "${module_name}/scripts"
  # source_prefix set at top
```

- Set the destination

```
"${script_path}/${name}":
```

- Set the mode 0755

Cron/Init Status

- Variables to ensure cron or init if want_* is true

```
$ensure_cron = $want_cron ? {
true    => 'present',   default => 'absent',}
# same for ensure_init
```

- Variables to set the location if want_* is true

```
$source_cron = $want_cron ? {
true => template("${source_prefix}/${name}.cron"),
 default => undef,  }
```

- Init location

```
true => template("${source_prefix}/${name}.init"),
```

Cron/Init Status

- Note that cron source is script_name.cron

```
true => template("${source_prefix}/${name}.cron"),
```

- Init source is script_name.init

```
true => template("${source_prefix}/${name}.init"),
```

Cron/Init Existence

- If a cron or init is not set, it should not exist
- Think cron script on master/slave & promotion

```
    file {
        "${cron_path}/${name}":
            ensure => $ensure_cron,
            mode => '0644',
            content => $source_cron;

        "${init_path}/${name}":
            ensure => $ensure_init,
            mode => '0755',
            content => $source_init;
    }
}
```

Script Considerations

- A script may need a particular database/user
- Instantiate ::script, ::database, ::grant
  - Parameters for script class
- Or make a new class
  - Instantiate ::script, ::database, ::grant in the class
  - Fewer lines in the actual instantiation

Script Considerations

- We made a new class for checksums
- Parameters are variables
- Adding a new template
- Style difference

Checksum Class: variables

- want_cron
  - Duplicate of script class
  - Denormalized
- Variables for the checksum script
  - chunk_size_limit
  - ignore_tables
  - password

Checksum Script

```
file {
  '/usr/local/bin/mysql-checksum.sh':
  content =>
template("${module_name}/mysql-checksum.sh.erb"),
  mode    => '0755';  }
```

- Template manages a lockfile and runs the script
- Some hard-coded params like ignore_db
  - mysql, checksum db, I_S, P_S

want_cron

- Same as what's in ::script
- Either create it or make sure the cron is absent
- Cron script name is hard-coded
  - Tradeoff
  - Could be a variable, but why?

New Class vs. ::script Tradeoffs

- Can require/set up ::database
- Can require/set up user with ::grant
- Maybe the script class should be more flexible?
  - Checksums are our most complex script

More Complex Cases

- More than one instance per server
- We do it for backups
- Does not call ::server class, similar to it
- Does call ::settings, ::client classes

Class Reverse Engineered

- From what we had in place
- /etc/sysconfig/mysql-backup-clusters file
  - Array of instance names
- Array is source of truth
- Array populated by instance variable

Class Reverse Engineered

- Local sockets, no ports
- Backup scripts use the array
  - Scripts are same on all servers
  - Not templates

Per-Cluster Variables

- Pass the array to a subclass
- ::cluster subclass
  - uses $title for each instance name
- Array might be [“foo”, “bar”, “baz”]
- Subclass is instantiated 3 times

Cluster name “foo”

/data/foo = datadir

/data/foo/foo.cnf = config file, includes password

/var/lib/mysql/foo.sock = socket file

/etc/init.d/mysqld-foo = init script

Cluster name “foo”

/var/run/mysqld/foo.pid = pid script

.bash_profile is script to make aliases mysql-foo:

mysql --defaults-file=/data/foo/foo.cnf

-S /var/lib/mysql/foo.sock

Backup Locations / crons

/data/backups/foo

/data/backups/foo/sqldumps

/data/backups/foo/rawcopies

/etc/cron.d/foo.cron – from a scripts directory

Questions/Comments/Feedback?

Slides:
http://bit.ly/puppet-mysql-slides

Puppet Module:
https://github.com/mozilla-it/puppet-mysql
Apache license