ECE 495/595 – Web Architectures/Cloud Computing

Module 5, Lecture 2: Web Application Security – Authentication

Christopher C. Lamb

University of New Mexico

What is Capistrano?

Capistrano is a distributed system management framework.

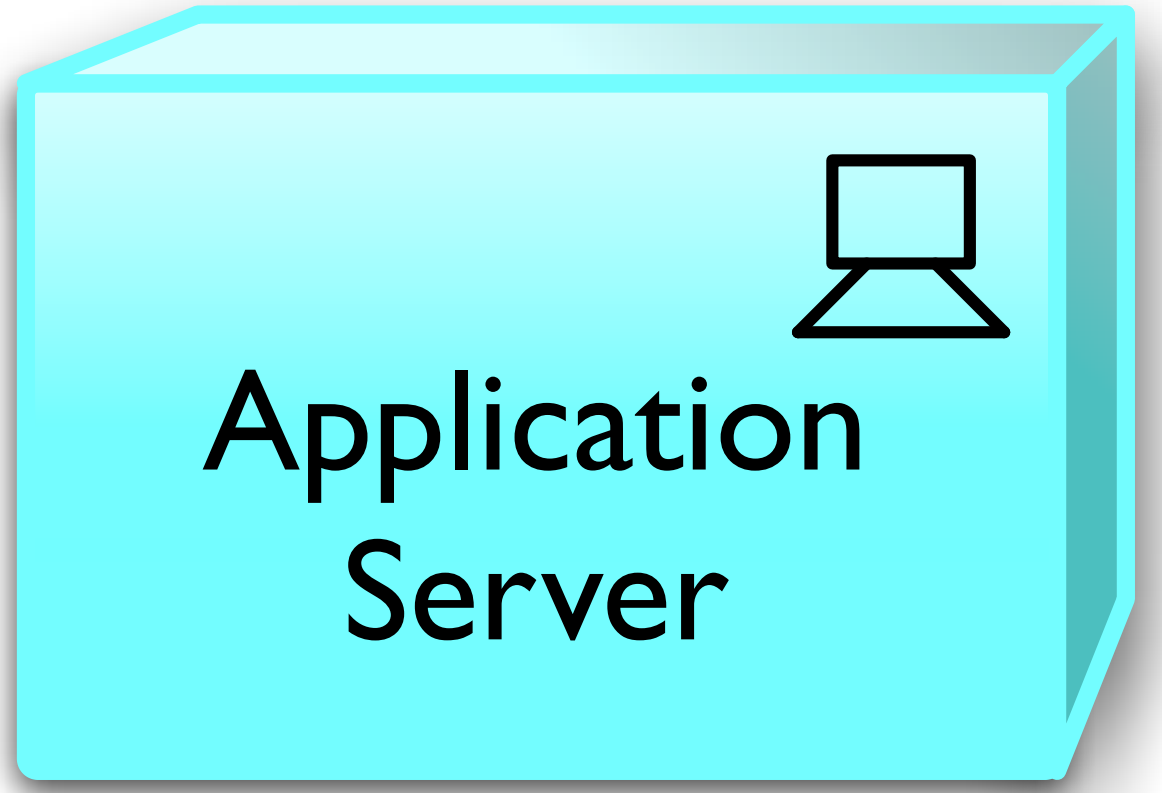
It supports:

- Distributed deployments.
- Command distribution.
- General large-scale system management.

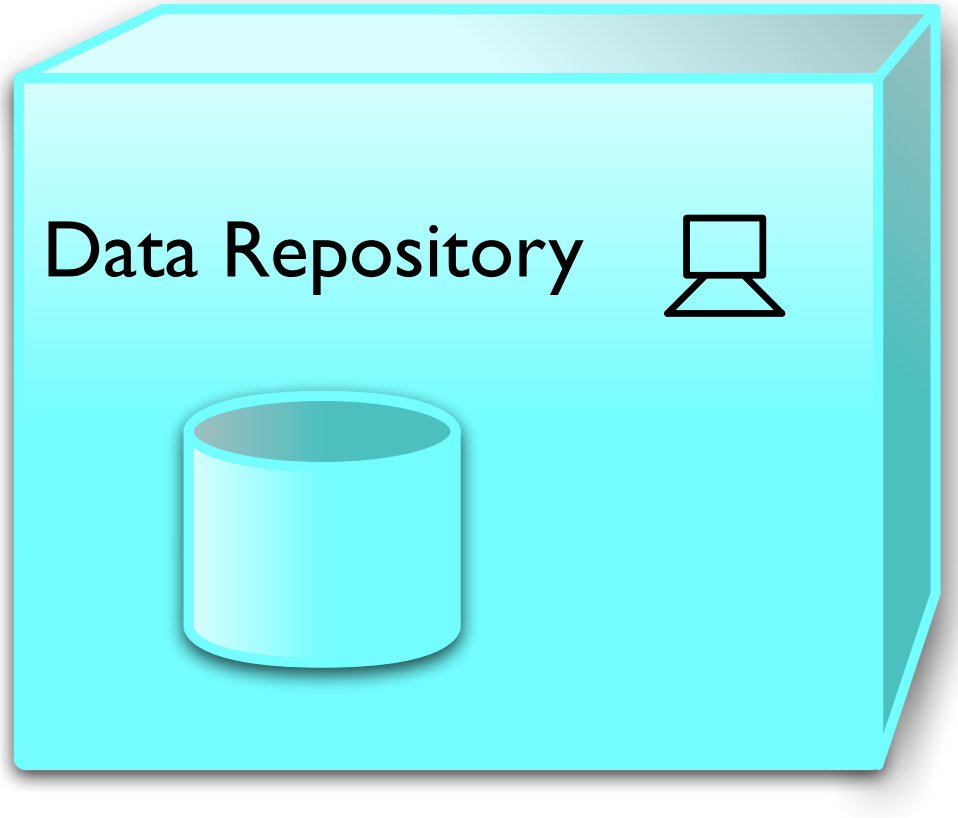
So what?

Starting out...

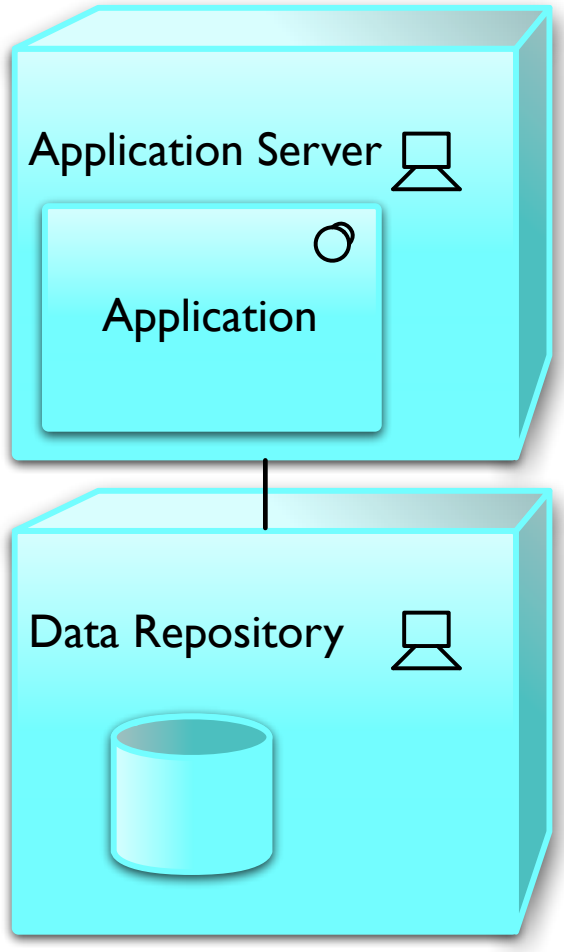
...where to put it?



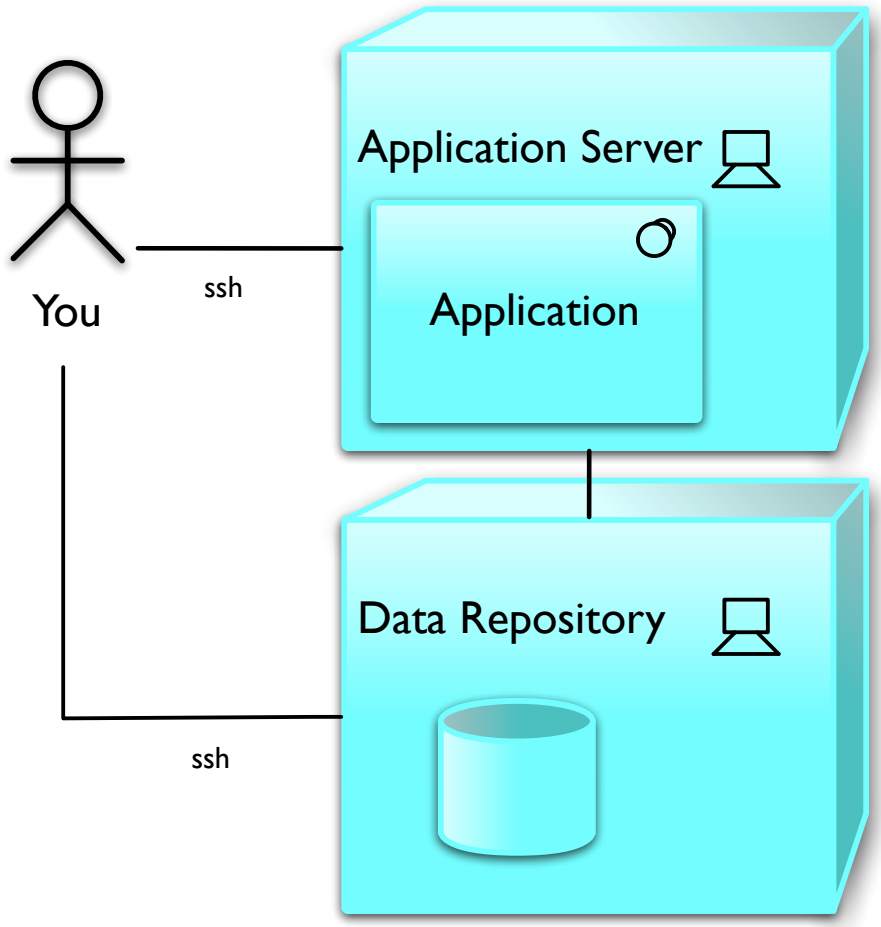
...and how to store stuff?



...now linking it together...



...and this is how you manage it.



Is this sufficient?

Well, probably okay for:

- School
- Departments
- Small organizations

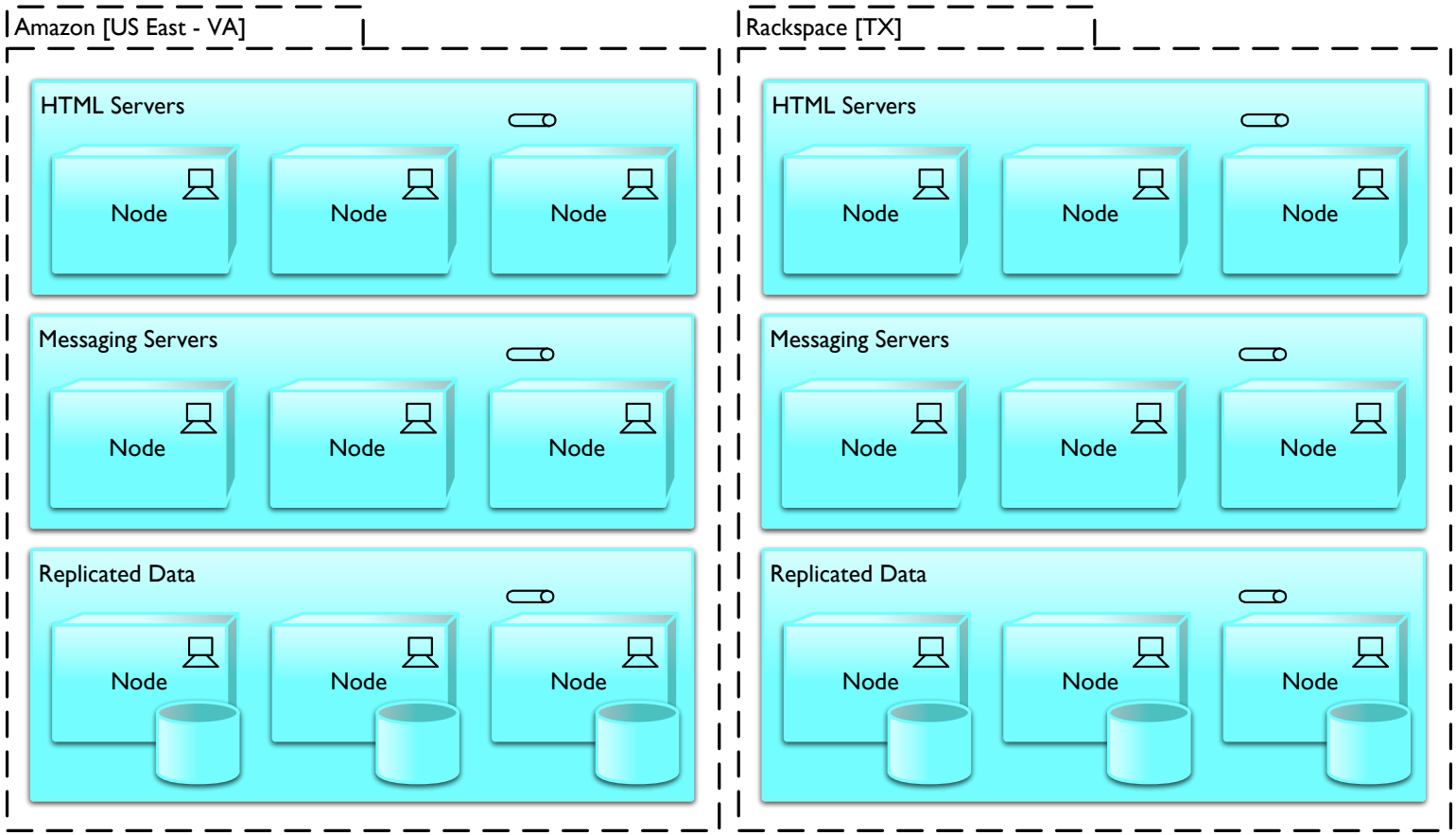
But honestly, not very real world. Systems with any kind of availability requirements or volume usually have:

- More systems
- Specialized systems
- More providers

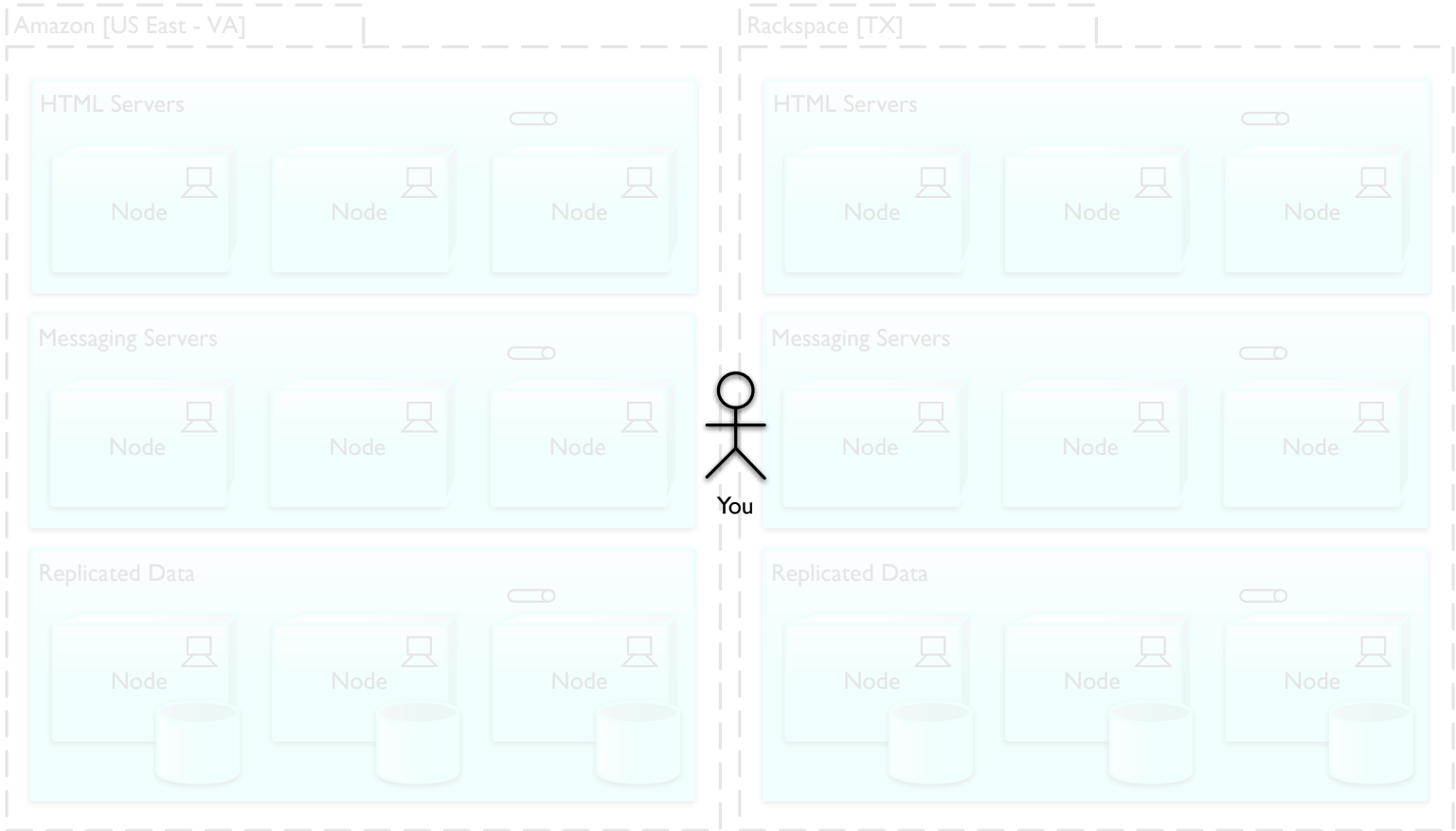
Really? Well, why not?

THE UNIVERSITY of NEW MEXICO

Typical small deployment...



...and you're responsible for it.



So how do you do it?

Well, you don't want to log into each system to administer manually. And why is that?

- **Scalability** You don't scale. Sorry.
- **Reproducibility** It's very difficult to recreate the same sequence of actions and configurations across multiple systems, so you'll generally script it. Which is what Capistrano does, in a distributed way.
- **Human Error** You make lots of mistakes too. If you can tell a machine what to do, it'll do a better job than you can.

So figure out how to tell machines to configure themselves. Or, use a system built by somebody else to tell machines to configure themselves, like Capistrano.

THE UNIVERSITY of NEW MEXICO

What you need to do first...

Capistrano can help you deploy applications, but it needs systems first.

You need to have systems set up and configured for use.

- **Chef, Puppet**
- **Vagrant**

They need to have supporting infrastructure installed, like **ruby**, **capistrano**, **bundler**, **rvm**, and **git**.

Setup isn't turnkey, but future use of Capistrano is.

Starting Out

Now, install Capistrano on your workstation:

```
$ gem install capistrano
```

Or use bundler, with a gem file:

```
source :rubygems
gem 'capistrano'
```

Now initialize the project:

```
$ cd <project directory>
$ capify .
```

This creates, in the project directory, a `config` directory containing a `deploy.rb` file, and a file in the project root named `Capfile`.

How do these files work?

Capfile:

```
load 'deploy'
load 'config/deploy'
```

config/deploy.rb:

```
set :application, "set your application name here"
set :repository,  "set your repository location here"

set :scm, :subversion

role :web, "your web-server here"
role :app, "your app-server here"
role :db,  "your primary db-server here", :primary => true
role :db,  "your slave db-server here"
```


What does this give you?

```
$ cap -T
cap deploy                # Deploys your project.
cap deploy:check          # Test deployment dependencies.
cap deploy:cleanup        # Clean up old releases.
cap deploy:cold           # Deploys and starts a `cold' application.
cap deploy:create_symlink # Updates the symlink to the most recently deployed...
cap deploy:migrate        # Run the migrate rake task.
cap deploy:migrations     # Deploy and run pending migrations.
cap deploy:pending        # Displays the commits since your last deploy.
cap deploy:pending:diff   # Displays the `diff' since your last deploy.
cap deploy:restart        # Blank task exists as a hook into which to install...
cap deploy:rollback       # Rolls back to a previous version and restarts.
cap deploy:rollback:code  # Rolls back to the previously deployed version.
cap deploy:setup          # Prepares one or more servers for deployment.
cap deploy:start          # Blank task exists as a hook into which to install...
cap deploy:stop           # Blank task exists as a hook into which to install...
cap deploy:symlink        # Deprecated API.
cap deploy:update         # Copies your project and updates the symlink.
cap deploy:update_code    # Copies your project to the remote servers.
cap deploy:upload         # Copy files to the currently deployed version.
cap deploy:web:disable    # Present a maintenance page to visitors.
cap deploy:web:enable     # Makes the application web-accessible again.
cap invoke                # Invoke a single command on the remote servers.
cap shell                 # Begin an interactive Capistrano session.
```

THE UNIVERSITY of NEW MEXICO

Capistrano is Biased

Nice! But Capistrano was developed expressly to deploy distributed Rails applications and carries baggage around from this heritage.

Unless you're deploying Rails, you'll want to get around this — **railsless-deploy** to the rescue!

```
$ gem install railsless-deploy
```

Or add to your Gemfile:

```
...
gem 'railsless-deploy'
...
```

Other plugins include `bundler/capistrano` and `rvm-capistrano`.

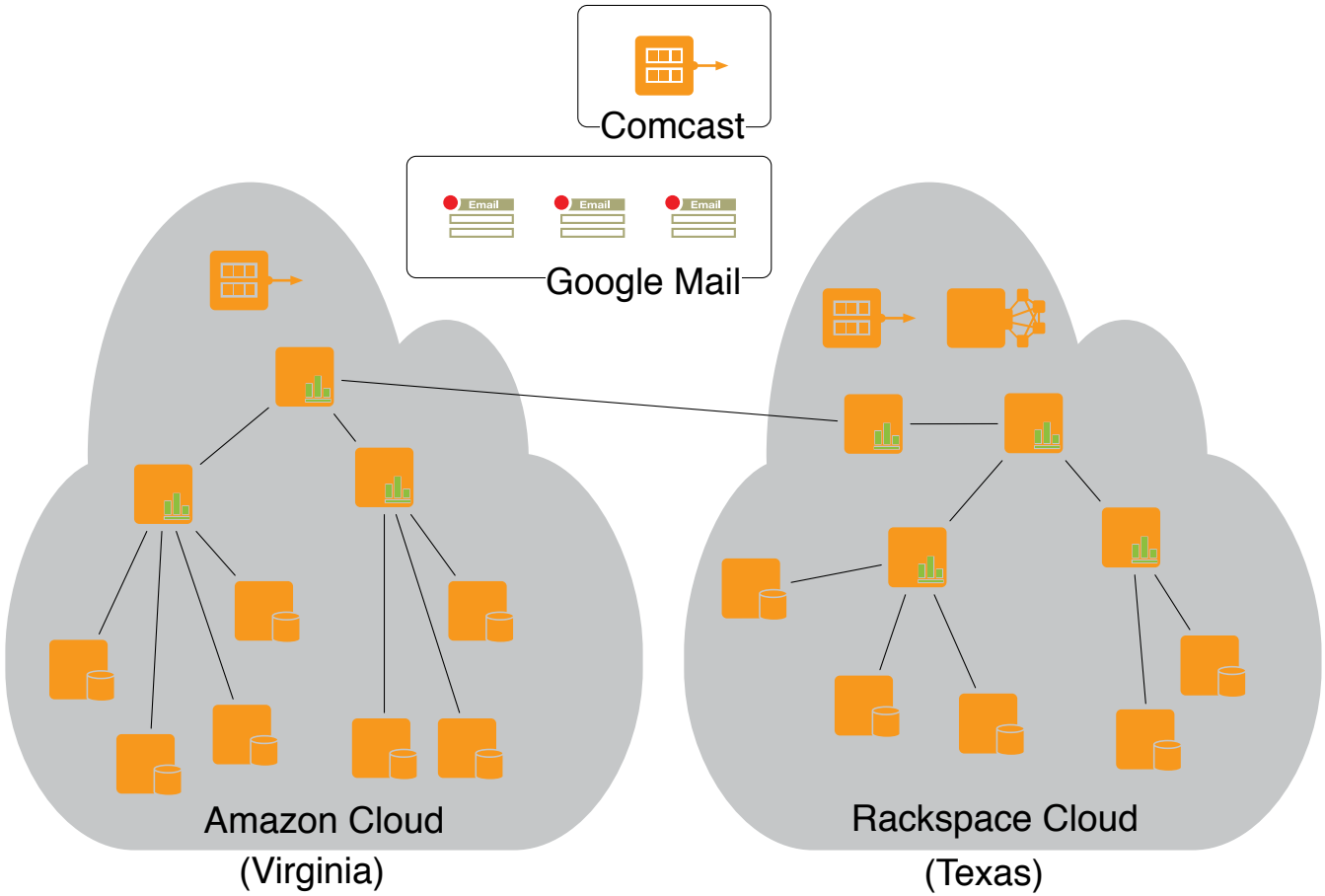
Other Useful Plug-ins

You'll want to pre-install **bundler** and **rvm** these on the system you're using. Why?

- **bundler** will ensure that your bundles are correctly installed. Remember you installed bundler in the base image? this also installs `bundler/capistrano`, which is what you use to integrate bundler.
- **rvm** is more than just a gem, so it's out of **bundler**'s league. A typical **rvm** install in your base image will suffice. After RVM is installed you can install `rvm-capistrano` with **bundler** or via the command line.

Now let's try this with a real system.

A Content Delivery Network



How do we support this?

This has 23 systems across 2 cloud infrastructure providers and integral external services.

What tools do we use?

- **Base Tooling** — Ruby, Bundler, RVM
- **Deployment** — Capistrano
- **SCM** — Git
- **HTTP Services** — S3, EC2, Rackspace Servers

Capfile

First, the Capfile:

```
require 'rubygems'           # Loading gem support
require 'railsless-deploy' # Loading railsless operation

# Loading vendor supplied plugins
Dir['vendor/gems/*/recipes/*.rb',
    'vendor/plugins/*/recipes/*.rb'].each do |plugin|
  load(plugin)
end

load 'config/deploy' # loading deployment tasks
```

Why are we altering this file, and not just the `config/deploy.rb` file?

We need to load the `railsless-deploy` support prior to the `config/deploy` targets.

config/deploy.rb

Now, the deployment configuration:

```
require 'bundler/capistrano' # Bundler support
require 'rvm/capistrano' # RVM support

require 'yaml' # YAML, for loading config info into S3
require 'aws-sdk' # Support for Amazon S3 APIs (and others)

set :rvm_ruby_string, '1.9.3' # The ruby version we'll use

ssh_options[:keys] = ['etc/pod.pem'] # Amazon key file

# Use GIT, and use this repository with this branch
set :scm, :git
set :repository,  'https://github.com/cclamb/overlay.git'
set :branch, 'final'

set :application, 'content-network' # Application name
```


config/deploy.rb

Now, the deployment configuration:

```
# The credentials to use; not maintained in Github
creds_file_name = 'etc/creds.yaml'

# The credentials are formatted in YAML.
creds = YAML::load File::open(creds_file_name)

# This is the format of the YAML file.
msg =<<EOF
Submitted credentials are:
  rackspace password: #{creds['rackspace']['password']}
  amazon access key: #{creds['amazon']['access_key']}
  amazon secret key: #{creds['amazon']['secret_key']}
EOF
puts msg

# Configuring SUDO and deploy locations.
set :use_sudo, false
set :deploy_to, '~'
```


config/deploy.rb

Setting credentials and keys:

```
# Setting some credentials for Rackspace.
set :user, 'overlay'
set :password, creds['rackspace']['password']

# Setting the keys for Amazon.
access_key = creds['amazon']['access_key']
secret_key = creds['amazon']['secret_key']
AWS.config \
  :access_key_id => access_key, \
  :secret_access_key => secret_key
```

Why do we have so many keys? Remember the PEM file?

The PEM file contains keys for **SSH**, the access and secret keys are used to authenticate into Amazon's infrastructure.

config/deploy.rb

What are the hosts?

```
role :nodes, '198.101.205.153', \
  '198.101.205.155', \
  '198.101.205.156', \
  '198.101.203.202', \
  '198.101.209.178', \
  '198.101.209.247', \
  '198.101.202.188', \
  '198.101.207.34', \
  '198.101.207.48', \
  '198.101.207.236', \
  'ec2-67-202-45-247.compute-1.amazonaws.com', \
  'ec2-23-20-43-173.compute-1.amazonaws.com', \
  'ec2-50-17-85-234.compute-1.amazonaws.com', \
  'ec2-23-22-68-94.compute-1.amazonaws.com', \
  'ec2-50-17-57-243.compute-1.amazonaws.com', \
  'ec2-50-16-141-176.compute-1.amazonaws.com', \
  'ec2-184-73-138-154.compute-1.amazonaws.com', \
  'ec2-107-20-47-98.compute-1.amazonaws.com', \
  'ec2-184-73-2-121.compute-1.amazonaws.com', \
  'ec2-23-22-144-216.compute-1.amazonaws.com'
```

THE UNIVERSITY of NEW MEXICO

config/deploy.rb

Some final configuration with S3:

```
# Push configuration information to S3
s3 = AWS::S3.new
config_bucket = s3.buckets[:chrislambistan_configuration]
config_bucket.clear!
obj = config_bucket.objects[:current]
obj.write :file => config_file_name
url = obj.url_for :read

# The logging bucket
bucket_name = 'chrislambistan_log'
```

Now we've finished configuration, we're ready for the targets.

config/deploy.rb

Namespaced Tasks:

```
namespace :nodes do

  task :start, :roles => :nodes do
    run "cd current ; bundle exec bin/main \"#{url}\" #{
        access_key} #{secret_key} #{bucket_name}"
  end

  task :monitor, :roles => :nodes do
    stream 'tail -f ./current/system.log'
  end

  task :stop, :roles => :nodes do
    run "cd current ; bundle exec bin/stop_overlay_component"
  end

end
```


Bringing it Together

Typical command-line use:

```
$ git add .
$ git commit -m 'deploying'
$ git push origin final
$ cap nodes:stop nodes:start nodes:monitor
```

In a more rigorous environment, you'd probably have a more stringent branching standard (e.g. merge branches to master, test, merge branches to stage, then deploy from stage).

This has two stages

- **Bootstrap** — Capistrano is key here. Distribute code and configuration data and start the system.
- **Run** — Nodes retrieve configuration and other information and begin operation.

Wrapping up

So what to remember?

Capistrano...

- ...is really useful for distributed deployments.
- ...does require some initial setup.
- ...is configured differently on different IaaS providers.
- ...is very flexible.
- ...but doesn't do everything!

That's it!

Questions?

