TECHNICAL WHITE PAPER

PRACTICAL GUIDE TO **CLOUD MIGRATION WITH CHEF AUTOMATE**

INTRODUCTION

The advent of cloud platforms has changed how applications are developed, and how infrastructure is deployed. By providing the ability to launch and scale environments on-demand, cloud platforms have allowed organizations to iterate faster than they could have imagined under previous operating models that required manual, tedious acquisition and configuration of inflexible data center infrastructure. Even so, migrating to the cloud comes with its own unique challenges that need to be addressed. This paper will start by articulating the challenges organizations face when beginning a cloud migration process, and provide examples of how Chef can help organizations meet these challenges, and take advantage of the myriad benefits of cloud adoption.

CHALLENGES:
What's Keeping Your Organization On-Prem?

Before you can realize the benefits of a cloud migration, it's important to address some of the reasons organizations choose to remain on premises. While migrating to the cloud can drastically impact your ability to quickly and efficiently deploy and scale environments, planning a successful migration is not without its challenges.

These challenges include:

Uncertainty over customer responsibilities. Cloud vendors provide tools to simplify management of environments, but it's often unclear where the vendor's responsibility ends, and the customer's begins. This uncertainty can delay migrations by introducing concerns over how to effectively mitigate risk. Even after responsibilities are identified and understood, organizations still need the ability to audit their estate to validate that their responsibilities have been met, and that environments are in compliance with their requirements.

Hybrid environment complexity. Most organizations, even among those that embrace cloud methodologies, still have at least some on premises footprint. It can be difficult to maintain hybrid infrastructure without also duplicating effort between environments, making processes difficult to scale across on-prem and cloud estates.

Maintaining deployment portability. Teams need to be trained on the tools of their selected cloud vendor or vendors, and it can be difficult to budget time and resources appropriately as these skills are being learned. This can be compounded when organizations are targeting multiple cloud providers, where tooling and configuration specifics might be unique to each.

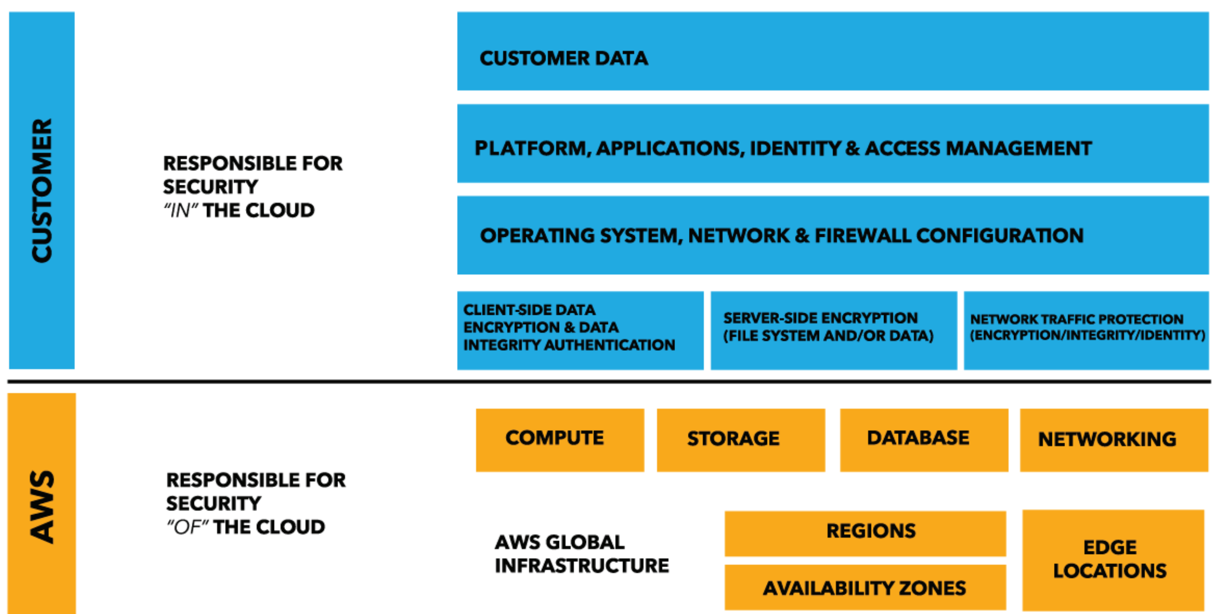
The sections that follow will address how Chef can help you meet these challenges without necessitating a top-to-bottom redesign of deployment procedures when migrating to a new platform. To achieve this, it's imperative to strive for Continuous Automation -- a repeatable workflow to detect, correct, and automate the configuration of all your systems.

Continuous Automation is critical to improving efficiency and reducing risk in cloud migration given the responsibility customers maintain for security in the cloud and the complexity involved in managing hybrid environments. This paper provides examples and guidance for using Chef Automate, Chef, and InSpec with complementary tools to detect misconfigurations on demand, correct configuration across hybrid environments, and automate deployments across any cloud and your entire estate.

UNDERSTAND YOUR RESPONSIBILITIES

Chef and the Shared Responsibility Model

In order to help you determine the scope of your responsibilities in the cloud, providers have created high-level guidance in the form of the "Shared Responsibility Model." These models aim to articulate where your cloud vendor's responsibilities end, and yours begin. The image below, from AWS, distinguishes between the two by defining AWS as responsible for the security "of" the cloud, whereas your responsibility is for security "in" the cloud. In other words, Amazon is responsible for maintaining and securing the infrastructure and tools they provide, and you're responsible ensuring that the applications and infrastructure are securely configured within that framework.



It's important to note that even within this separation, there is still some overlap in responsibilities to address. For example, while "storage" is listed among AWS's responsibilities in the above chart, you are still responsible for ensuring that the data you store is properly encrypted, and has appropriate access controls. AWS provides and maintains the tools for accomplishing these tasks, but it's your responsibility to put them into practice according to your organization's needs.

Responsibility	On-Prem	IaaS	PaaS	SaaS
Data classification & accountability	Cloud Customer	Cloud Customer	Cloud Customer	Cloud Customer
Client & end-point protection	Cloud Customer	Cloud Customer	Cloud Customer	Cloud Customer / Cloud Provider
Identity & access management	Cloud Customer	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Customer / Cloud Provider
Application level controls	Cloud Customer	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Provider
Network controls	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Provider	Cloud Provider
Host infrastructure	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Provider	Cloud Provider
Physical security	Cloud Customer	Cloud Provider	Cloud Provider	Cloud Provider

Cloud Customer   Cloud Provider

A similar chart from Microsoft Azure (right) puts a finer point on some of these areas of overlap. Here, any of the boxes that have a single color represent the sole responsibility of either the customer or cloud provider, whereas boxes that contain both grey and blue indicate a combination of the two, as outlined in the storage example above. As organizations move towards more cloud-native PaaS and SaaS offerings, they're able to offload more security concerns to their cloud provider, but even in the rightmost column, we can see that there will always be areas that you will be responsible for configuring and validating, even within cloud-provided frameworks.

Validate Your Requirements with InSpec

Chef's software offerings are designed to help you manage your areas of responsibilities in a repeatable, platform-agnostic way. Chef accomplishes this by providing Domain Specific Languages (DSLs) that allow you to create code to detect deviations from security and configuration standards (InSpec) and correct any misconfigurations that arise (Chef). What makes Chef code portable is that it's declarative by design -- in other words, you're responsible for defining **what** needs to be configured, and Chef can dynamically determine **how** those configurations need to be applied wherever it's being run. Ultimately this means that cloud-provisioned instances can be validated and configured with the same code used on traditional bare metal or VM based machines in your datacenter.

For a simple example, consider the installation of NGINX, a popular webserver. You can detect whether it's installed on a system via InSpec, by writing a control that looks something like this:

```
control 'verify-nginx' do
  impact 1.0
  title 'Verify whether nginx is installed'
  describe package('nginx') do
    it { should be_installed }
  end
end
```

To actually install NGINX, Chef's package resource can be used in a similar fashion:

```
package 'nginx' do
  action :install
end
```

While this is a simple example, it encapsulates what makes tools like InSpec and Chef powerful. Regardless of what OS flavor you're running, or where your instances are deployed, the same code can be used to validate and configure your systems. While different environments will have differences between them that need to be accounted for, those aspects that are common between them don't need to be managed any differently. This becomes increasingly important as you migrate workloads into the cloud, as you only need to create automation for the net-new components of your design, and can avoid duplicating efforts that have already been automated on-prem.

MANAGE HYBRID ENVIRONMENTS

Chef Cloud Integrations

Everything that's been discussed so far can be applied to on-premises and cloud environments alike, but organizations looking to migrate to the cloud have opportunities and challenges that differ from their on-prem only counterparts. Cloud providers offer tools and features to assist your organization in iterating more quickly, but making effective use of them takes time and resources as your teams are trained up. In these scenarios, Chef further assists in cloud adoption by providing a variety of integrations that allow you to take advantage of the cloud's capabilities, without having to create workflows unique to each cloud or datacenter you manage. In this section, we'll look at some of these to give you insight into how you can prepare for managing your environments in the cloud.

Audit More than Just Servers with InSpec

A unique aspect of administering systems in the cloud is that you're no longer just managing servers and network devices; most cloud platforms provide managed offerings for everything from data storage to access control to load balancing. While using these tools can help ease the burden of creating and maintaining home-grown solutions from the ground up, you still need a way to validate that they're properly configured. InSpec has resources to help you do just that.

For example, networking configurations are typically handled by configurable abstractions rather than directly modified on a switch or router. In AWS, customers can create Virtual Private Clouds (VPCs) which define, among other things, an isolated private network for an environment. In InSpec, you can use the aws_vpc resource to define your expectations for such the same way you might evaluate the content of a running VM.

```
describe aws_vpc('vpc-12345678') do
  it { should exist }
  it { should be_default }
  its('cidr_block') { should cmp '10.0.0.0/16' }
  its('state') { should eq 'available' }
end
```

Even when you *are* running virtual machines within the cloud, you might want to validate some of that instance's metadata which isn't always directly available from within the instance itself. Again, InSpec can be used here to capture that information thorough similar cloud-specific resource. Below is an example of some VM validations in Microsoft Azure.

```
describe azure_virtual_machine(group_name: 'Inspec-Azure', name:
'Windows-Internal-VM')
do
  its('os_type') { should eq 'Windows' }
  it { should have_boot_diagnostics }
  its('location') { should eq 'westeurope' }
  it { should have_data_disks }
  its('vm_size') { should eq 'Standard_DS2_v2' }
end
```

Both examples reference data that can typically only be collected via your cloud provider's GUI or API, but expressed in a way that can be combined with server-specific controls in a single language, and aggregated in a single dashboard with Chef Automate.

Cloud Configuration as Code with Chef

These same sets of PaaS and cloud-native tooling present unique challenges for automating configuration management, as with security validation. To address this, Chef has numerous integrations to allow you to configure these services the same way that InSpec helps you validate them. As with InSpec's cloud resources, Chef provides resources specific to common cloud workflows to assist in automating their management.

One popular example is how data storage is managed in the cloud. Most cloud providers offer a native storage solution that allows for creating shared storage resources with built in RBAC capabilities without the additional overhead of maintaining traditional network storage devices. Chef helps support this model by providing cloud-specific resources for managing those objects with the same syntax you use to manage software configurations on the servers themselves. Below is an example using Azure's Storage Containers.

```
microsoft_azure_storage_account 'my-account' do
  management_certificate microsoft_azure['management_certificate']
  subscription_id microsoft_azure['subscription_id']
  location 'West US'
  action :create
end

microsoft_azure_storage_container 'my-container' do
  storage_account 'my-account'
  access_key microsoft_azure['access_key']
  action :create
end
```

The code above will first create a storage account, called 'my-account', followed by a storage container called 'my-container', which will be created in said account. Similar resources exist for AWS (aws_s3_bucket) and GCP (gstorage_bucket), each with configurable parameters to capture the associated cloud's unique tunables. Analogous resources exist for similar cloud-specific PaaS offerings including management tools for DNS, IAM, Load Balancing, Databases, and more.

Dynamic Configuration: Multiple Platforms, One Codebase

Another key benefit of defining configuration as code is that you can create dynamic behavior based on system profiling data, collected by a chef component called ohai. Ohai is run every time a node starts a chef-client run, and collects information about that node that can be used to trigger conditional behavior.

A simple illustration of where you might need this ability is when packages are named differently in different operating systems. For example, the webserver software "apache" can be run on both Debian and Redhat based OSes, but the package names differ between these operating systems. Generally, in debian, the package is called "apache2", whereas in redhat, it's called "httpd". Chef's package resource can be run on either system, but you'll still need to account for their different naming conventions in your code. Since ohai is already capturing your OS information automatically, you can use that data to conditionally alter your execution without having to maintain separate codebases, as illustrated in the example below.

```
if node['platform_family'] == "debian"
  node.default['apache']['package'] = "apache2"
elsif node['platform_family'] == "rhel"
  node.default['apache']['package'] = "httpd"
end

package node['apache']['package'] do
  action :install
end
```

In the above example, the "package" attribute is set based on what "platform_family" is detected by ohai, and the package resource will substitute the appropriate value for "node['apache']['package']". Ohai also collects data about what cloud provider, if any, your node was deployed within, allowing us to use cloud-specific input as a condition for execution as well. Consider what happens if your organization has infrastructure deployed on premises and in AWS, and you maintain load balancers in both environments. As with the apache example, you can use the ohai data collected during a chef client run to inform how you configure those systems dynamically. In the example below, a "load_balancer" cookbook has been created with an "haproxy" recipe for configuring on-prem systems, and a "elb" recipe to configure an Elastic Load Balancer on AWS. Rather than add those recipes directly to the run list of associated systems, you can instead have your "default" recipe decide which of these more specific recipes to include based on the information collected by ohai.

```
if node['cloud']['provider'] == "ec2"
  include_recipe "load_balancer::elb"
else
  include_recipe "load_balancer::haproxy"
end
```

Defining that logic within your recipe allows the same "default" recipe to be run regardless of the underlying environment, and Chef will take the appropriate action in each based on your conditional statement. The benefits of this capability are twofold. Because the logic is controlled by code, misconfiguration due to human error is easier to avoid. Because it allows the same recipe to be applied to disparate systems, it simplifies your overall configuration by having environmental differences evaluated dynamically, rather than in manually-defined bespoke permutations.

MAINTAIN DEPLOYMENT PORTABILITY FOR ANY CLOUD

Provisioning - Consistent Deployment in Any Cloud

One of the primary benefits of moving to the cloud is the ability to provision infrastructure on demand. This can range from development and QA teams having the ability to self-serve environments as needed, operations teams having the ability to dynamically scale environments, and the removal of procurement as a bottleneck for all of the above. Chef helps organizations ensure that as these instances, services, and environments are created, they're configured consistently regardless of which team initiates the request, or which environments are being updated.

Testing in the cloud with Test Kitchen

The Chef Development Kit (ChefDK) comes packaged with a testing harness called Test Kitchen. At a high level, Test Kitchen provides a repeatable workflow for instantiating, configuring, verifying, and destroying ephemeral infrastructure for easy testability at velocity and scale. What's better, all four of those steps can be accomplished by running a single command: "kitchen test".

As you might expect, when running Test Kitchen, Chef is the default tool for applying configurations, whereas InSpec is the default tool for verification. By contrast, when it comes to provisioning and destroying those instances, Test Kitchen provides the flexibility to do so in a variety of ways. Test Kitchen has a variety of configurable drivers for launching these instances on your local machines (e.g Vagrant, Docker), into on-premises virtualization (e.g. VMWare), or into your cloud environment (e.g. AWS, Microsoft Azure, GCP). A configuration file called kitchen.yml controls which of these drivers should be used, and is where you can define driver-specific behavior for your cloud of choice. Here's an example using the kitchen-ec2 driver, which allows you to launch your testing instances in AWS.

```
driver:
  name: ec2
  security_group_ids: ["sg-1a2b3c4d"]
  region: us-west-2
  availability_zone: b
  subnet_id: subnet-6e5d4c3b
  iam_profile_name: chef-client
  instance_type: m3.medium
  associate_public_ip: true
```

In the driver config, you define the cloud-specific settings for your instance. In the above example, the driver section defines security groups, region, subnet, etc -- all of which are specific to AWS. The remainder of the config contains information independent of the selected cloud, such as operating systems, transports, and configuration details. The advantage to this configuration is that, regardless of how or where your instance is instantiated, the process for doing so remains the same:

```
kitchen test
```

Server Provisioning with Knife Cloud Plugins

Another core component of the ChefDK is a multipurpose utility called knife. Knife is the CLI used to perform most day-to-day administrative tasks in a Chef environment, and is also a tool that can be used to create and configure new instances in the cloud. Much the same way that test kitchen allows us to create cloud-specific configuration files, knife allows us to define both the cloud-specific and chef-specific information of any newly-created instance in a single command. Here are some examples on AWS, Azure, and GCP.

AWS - knife-ec2

```
$ knife ec2 server create
  -I ami-cd0fd6be
  -f t2.micro
  --aws-access-key-id 'Your AWS Access Key ID'
  --aws-secret-access-key "Your AWS Secret Access Key"
  -x myuser -P mypassword
  -N web-server-1 -E development -r 'role[webserver]'
```

Azure - knife-azure

```
knife azurerm server create
  --azure-resource-group-name MyResourceGrpName
  --azure-vm-name my-new-vm-name
  --azure-service-location 'westus'
  --azure-image-reference-publisher 'MicrosoftWindowsServer'
  --azure-image-reference-offer 'WindowsServer'
  --azure-image-reference-sku '2012-R2-Datacenter'
  --azure-image-reference-version 'latest'
  -x myuser -P mypassword
  -N web-server-1 -E development -r 'role[webserver]'
```

GCP - knife google

```
knife google server create
  --gce-image centos-7-v20160219
  --gce-machine-type n1-standard-2
  --gce-public-ip ephemeral
  -x myuser -P mypassword
  -N web-server-1 -E development -r 'role[webserver]'
```

In all three of the above examples, the first group of flags (e.g. `--aws-access-key-id`, `--azure-service-location`, `--gce-machine-type`) are specific to the cloud providers themselves.

The penultimate line of each contains connection details, in this case username (`-x myuser`) and password (`-P mypassword`)

The final line contains chef specific information:

- `N web-server-1` - Sets the 'node name', or unique identifier on the Chef Server.
- `E development` - Sets the chef environment, which can be used to organize classes of machines.
- `r 'role[webserver]'` - Sets the 'run list', or the Chef recipes and/or roles that should be applied to the instance once it's created.

In all three examples, this single command does a few discrete things:

- Launches a new instance in the specified cloud.
- Installs the Chef Client.
- Registers the instance with a Chef Server, with the node name and environment specified.
- Applies the recipes specified in the run list.

When complete, you have a newly created instance, already configured with the appropriate data defined in your run list.

Environment Provisioning with Chef & Terraform

The tools covered in this section thus far have focused primarily on the instantiation of individual instances. A fully realized environment will likely also require defining associated network, storage, and any other cloud-specific offerings that need to be configured alongside the instances being managed by Chef. To address the complexity of defining and managing that entire ecosystem, vendors have provided templating languages to provide a consistent way to provision environments end-to-end (e.g. AWS Cloudformation, Azure ARM). While Chef can be used within these frameworks, each format is only usable within its associated cloud, and cannot be easily ported between providers without maintaining separate templates for each.

Hashicorp Terraform is a provisioning tool that, like Chef, provides a declarative configuration language designed for cloud-agnostic abstraction. At a high level, Terraform combines a library of providers, responsible for defining interactions with specific clouds' APIs, with provisioners, responsible for defining how infrastructure should be configured once they're launched. As with the Test Kitchen example covered earlier, this allows for combining components into a single configuration file, and launching a full environment stack with a single command: "terraform apply".

```
# ...
# Configure the AWS Provider
provider "aws" {
  access_key = "${var.aws_access_key}"
  secret_key = "${var.aws_secret_key}"
  region     = "us-east-1"
}

# Create a web server
resource "aws_instance" "web" {
  # ...
  provisioner "chef" {
    environment     = "production"
    run_list        = ["webserver::default"]
    node_name       = "webserver1"
    server_url      = "https://chef.company.com/organizations/org1"
    version         = "12.4.1"
  }
}
# ...
```

Above is a simplified snippet of a Terraform configuration illustrating some of these integrations in action. This example will launch instances into AWS, and configure them with Chef. The "provider" section defines any AWS-specific information, similar to the "driver" section of the Test Kitchen example. The "provisioner" section, by contrast, defines chef-specific information, including what version of Chef to install, what Chef Server it should be communicating with, and what configuration steps should be taken once it's created.

Chef Automate in the Cloud - Visualize Your Automation

Once you start managing environments at scale, it becomes increasingly essential to have a way to maintain visibility into configuration state and system health across your estate, whether it exists entirely in a single cloud, across multiple clouds, or across a hybrid on-prem/cloud environment. Chef Automate is a platform that provides a single pane of glass into every system you manage, with change and audit history aggregated in a single location.

For customers of AWS and Microsoft Azure, getting started with Chef Automate is easier than ever! While Automate is packaged for easy installation anywhere, both AWS and Azure provide pre-configured Marketplace images that can be deployed directly into your cloud environment with a single click of a button.

AWS users also have the option of deploying Opsworks for Chef Automate (OWCA), providing a fully managed instance of Chef Automate, configurable from your AWS Management Console. With OWCA, day-to-day administrative tasks, like version upgrades and regular backups, are handled automatically, with user-friendly tools for configuring frequency and retention periods within the dashboard.

REVIEW

This paper discusses how Chef supports the cloud shared responsibility model by providing resources and integrations designed to help you automate management of your cloud environments with the same tools and workflows used on premises. Key points covered include:

- Customers are responsible for securing their applications and data, even when services surrounding them are secured by their cloud provider.
- InSpec and Chef are designed with automation and repeatability in mind, making it easier for organizations to adapt on-prem workflows to the cloud.
- Chef Automate provides visibility into current state as well as change and audit history to ensure organizations can easily and effectively assess change across environments and departments.
- Chef's tools and platforms provide built-in integrations with popular cloud providers to ensure a consistent workflow on-prem and in the cloud.
- InSpec and Chef can be used to manage servers and cloud-native PaaS offerings alike through a consistent syntax for detecting and correcting issues respectively.
- Cloud Marketplace images and Opsworks for Chef Automate provide ways to reduce friction for organizations looking to deploy Automate in their environments.

If your organization is undergoing a cloud migration, or if you're simply looking to improve existing processes, Chef is here to help! With a variety of cloud-focused resources, a vibrant community of open source contributors, and a wealth of experience driving organizational transformation for thousands of companies, take advantage of Chef's automation capabilities to move to the cloud with speed and confidence.

RESOURCES

Learning

Chef on AWS: https://learn.chef.io/tracks/chef-on-aws#/

Chef on Microsoft Azure: https://learn.chef.io/tracks/chef-on-azure#/

Documentation

Test Kitchen: https://docs.chef.io/kitchen.html

Knife Plugins: https://docs.chef.io/plugin_knife.html

InSpec AWS Resources: https://www.inspec.io/docs/reference/resources/#aws-resources

InSpec Azure Resources: https://www.inspec.io/docs/reference/resources/#azure-resources

Chef AWS Cookbook: https://supermarket.chef.io/cookbooks/aws

Chef Azure Cookbook: https://supermarket.chef.io/cookbooks/microsoft_azure

Chef GCP Cookbooks: https://supermarket.chef.io/users/googlecloudplatform

Related Reading

Cloud Migration Solutions Page: https://www.chef.io/solutions/cloud-migration/

Webinar: Successfully Migrate to the Cloud with Chef and AWS: https://blog.chef.io/2018/02/08/successfully-migrate-cloud-chef-aws/

Blog: InSpec 2.0 Cloud Resources Mini-Tutorial: https://blog.chef.io/2018/02/20/inspec-2-0-cloud-resources-mini-tutorial/