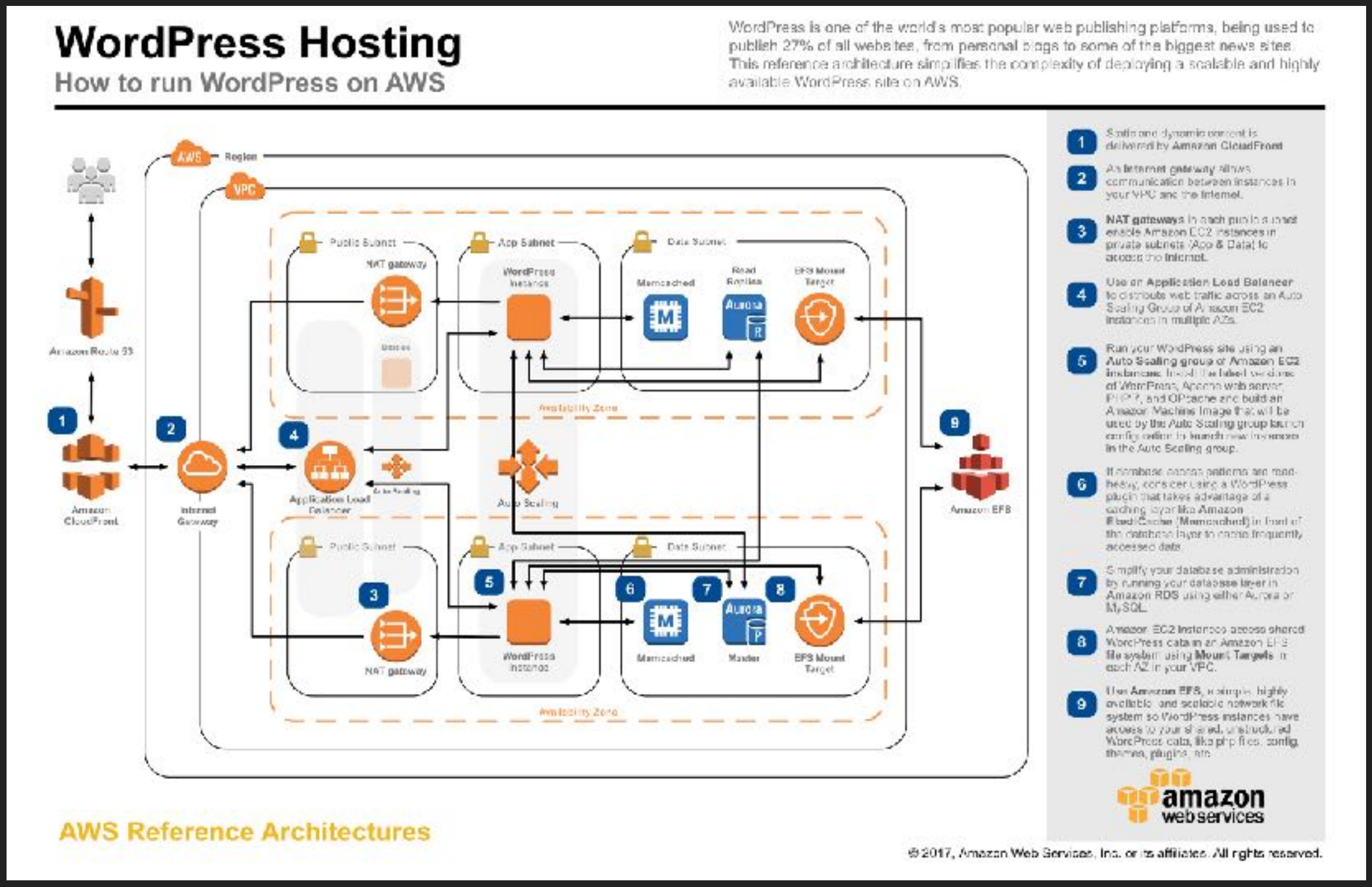
PATRICK TWOHIG

HCL VS CF

USE CASES FOR INFRASTRUCTURE AS CODE

- Migration to cloud-based infrastructure from traditional hard servers.
- Automated deployment, configuration setup for multiple environments. (development, staging, testing, and production).
- Continuous integration and deployment. Hooks to systems such as Jenkins, CircleCI etc. Rapid delivery, testing, and releases.
- Replication of systems in multiple data centers, locations, and availability zones.
- Streamlining workflow between development and operations by allowing smooth handover processes
- Rapid prototyping and design of infrastructure without needing to take valuable cycles from sysadmin/operations team members.
- Providing multi tenancy for a service at the infrastructure level, as opposed to an application level.

TYPICAL CLOUD INFRASTRUCTURE EXAMPLE



Official AWS Example setup for WordPress

TYPICAL CLOUD INFRASTRUCTURE CONFIGURATION STEPS

- Create all resources (Subnets, Routing Tables, EC2/Server Instances)
- Configure of services, (eg connecting web server to the database)
- Apply and test security restrictions (IP whitelists, firewalls, reverse proxies etc)
- Setup and configure of access permissions, granting/revoking access to specific internal organization users.
- Connect the backend infrastructure to VPN allowing sysadmin/operations team to access the live instances.
- Repeat this process several times over, to work out all the kinks in the system.

INFRASTRUCTURE AS CODE BENEFITS AND OVERVIEW

- Engineers define infrastructure and related configuration in scripts
- Scripts are interpreted by the tools and applied to to the infrastructure
- State of infrastructure is managed internally by the tools, allowing for future changes to be made incrementally.
- Greatly reduces the need to manually configure resources using traditional tools or methods.
- Similar in spirit to Chef, Puppet, etc. but for infrastructure and not existing single instance machines.

COMMONALITIES BETWEEN CLOUDFORMATION AND TERRAFORM

- Ability to control certain aspects of infrastructure using variables/macro substitution
  - Region-specific settings such as AMIs (Amazon Machine Images).
  - Customer-specific settings such as domain/subdomain names.
- Readily integrate with popular continuous integration systems allowing for automated deployment and configuration.
  - Jenkins
  - CircleCI
- Tools and editor support
  - IntelliJ
  - Eclipse
  - Vim

COMMONALITIES BETWEEN CLOUDFORMATION AND TERRAFORM

- Production Quality
  - Well Documented with plenty of resources, books, and references available
  - Used in real-world applications on non-trivial systems
  - Commercial and community support available
  - Active developer and user communities
- The ability to read outputs from the managed infrastructure, including:
  - ARNs of various resources
  - IP addresses of EC2 instances
  - Resource IDs etc

BASIC OPERATION FROM A USER PERSPECTIVE

- Engineers codify resources into one or more files defining the underlying cloud infrastructure.
- The framework tracks state information, such that it may incrementally sync changes in code to changes in infrastructure
  - CloudFormation: “Stacks”
  - Terraform: “State” (or “State Files”)
- User applies changes, and the affected resources are created, destroyed, or modified depending on the particular scenario.
- Post-run, downstream processes may read information back out of the state for further processing if necessary.

BASIC OPERATION INTERNALS

- Each resource (such as a server, load balancer, or NAT) is defined as an object with properties and dependencies in the definitions scripts.
- The system generates a graph of all resources in memory, ensuring there are no circular dependencies.
- Once generated, the graph is compared to the state information and produces an execution plan.
- An attempt is made to apply the to the infrastructure, reporting any errors or misconfigurations along the way.

TERRAFORM: CORE CONCEPTS

- Developed by HashiCorp to solve infrastructure management issues.
- Code syntax is a proprietary language, called HCL resembling JSON with some enhancements.
- Definitions can span multiple scripts existing in a single directory.
- Functions exist for processing and text manipulation.
- HCL is non-procedural code.

TERRAFORM: CORE CONCEPTS

- Stores and tracks state information in a file, called a state file, which is may be stored in many locations.
- Best practices dictate that files are stored in places where they are globally accessible.
- Infrastructure is modified through AWS API calls dispatched from the machine running Terraform.

CODE SAMPLE: TERRAFORM

```
resource "aws_launch_configuration" “my_launch_config” {

  name          = "${var.name}-asg-lc"
  image_id      = "${data.aws_ami_ids.asg_ami.ids[0]}"
  instance_type =
    “${var.instance_types[data.aws_ami_ids.asg_ami.ids[0]]}"

  iam_instance_profile   = "${aws_iam_instance_profile.ecs.name}"
  associate_public_ip_address = false

  lifecycle {
    create_before_destroy = true
  }

  user_data = <<USER_DATA
#!/bin/bash
# Set the cluster
echo \
  ECS_CLUSTER=${aws_ecs_cluster.cluster.name} >>\
  /etc/ecs/ecs.config
USER_DATA

  # Security group
  security_groups = [“${aws_security_group.my_sg.id}”]
  key_name        = “${var.my_key_pair}”

}
```

```
resource "aws_autoscaling_group" "asg" {

  availability_zones      = [
   "${data.aws_availability_zones.available.names[1]}",
   “${data.aws_availability_zones.available.names[2]}"
  ]

  name                    = "${var.name}-asg"
  max_size                = "${var.asg_max_size}"
  min_size                = "${var.asg_min_size}"
  desired_capacity        = "${var.asg_desired}"
  default_cooldown        = "${var.default_cooldown}"

  force_delete            = true
  launch_configuration    = “$
{aws_launch_configuration.my_launch_config.name}”

  load_balancers = ["${aws_elb.service_elb.name}"]

  lifecycle {
      create_before_destroy = true
  }

  depends_on = [
  “aws_launch_configuration.asg_launch_configuration"
  ]

  vpc_zone_identifier = [
   "${aws_subnet.private_subnet_primary.id}",
   “${aws_subnet.private_subnet_secondary.id}"
  ]

  tag {
    key                 = "Name"
    value               = "${var.name}-worker"
    propagate_at_launch = "true"
  }

}


}
```

CLOUD FORMATION: CORE CONCEPTS

- The “house brand” infrastructure as code solution provided by Amazon.
- State information is kept internally on AWS in what are called stacks.
- AWS determines the execution plan is on the remote end and then executed automatically.
- Errors are reported in the AWS control panel as they happen. Failures are automatically rolled-back.

CLOUD FORMATION: CORE CONCEPTS

- Template syntax is one of two languages: YAML or JSON.
- A limited subset of functions are available to perform functions such as text manipulation, extracting resource IDs, and processing parameters.
- There exists one stack per YAML/JSON file and sub stacks may be created or referenced therein.
- Practically speaking, sources must be copied to a global location (ie S3) where they can be processed as they are not run on the local machine.

CODE SAMPLE: CLOUD FORMATION

```yaml
AWSTemplateFormatVersion: "2010-09-09"

Description: Defines the Auto-Scaling Group for the SocialEngine ECS
Cluster

Parameters:

  # Required Parameters

  Cluster:
    Type: String
    Description: The ECS Cluster which will run the Service

  SecurityGroups:
    Type: CommaDelimitedList
    Description: The security groups for the EC2 Instance

  IamInstanceProfile:
    Type: String
    Description: The ECS Instance Profile

  ScriptStorage:
    Type: String
    Description: The ID of the Git Storage EFS Volume

  AvailabilityZones:
    Type: CommaDelimitedList
    Description: A list of availability zones in which to run the
instances

  Subnets:
    Type: CommaDelimitedList
    Description: A listing of subnets to use in this ASG

LaunchConfiguration:
  Type: AWS::AutoScaling::LaunchConfiguration
  Properties:
    AssociatePublicIpAddress: false
    EbsOptimized: !Ref EbsOptimized
    IamInstanceProfile: !Ref IamInstanceProfile
    ImageId: !Ref AMI
    InstanceType: !Ref InstanceType
    KeyName: !Ref KeyName
    SecurityGroups: !Ref SecurityGroups
    UserData:
      Fn::Base64:
        Fn::Sub: |
          #!/usr/bin/env bash
          # Configures the instance join to the Cluster
          echo ECS_CLUSTER=${Cluster} >> /etc/ecs/ecs.config

          # Install NFS Utilities
          yum install -y nfs-utils

          # Configures EFS to be used in this container

ASG:
  Type: AWS::AutoScaling::AutoScalingGroup
  DependsOn:
    - LaunchConfiguration
  Properties:
    AvailabilityZones: !Ref AvailabilityZones
    Cooldown: !Ref Cooldown
    MinSize: !Ref MinSize
    MaxSize: !Ref MaxSize
    DesiredCapacity: !Ref DesiredCapacity
    HealthCheckType: EC2
    LaunchConfigurationName: !Ref LaunchConfiguration
    VPCZoneIdentifier: !Ref Subnets
    Tags:
      - Key: "Name"
        Value: !Sub ${Cluster} Worker
        PropagateAtLaunch: true
```

DEALING WITH EXISTING INFRASTRUCTURE

- CloudFormation does not readily import existing infrastructure into stacks.
- CloudFormation is not very tolerant of manual changes after the fact.
- Terraform allows for importing of existing resources into a particular state file.
- Terraform tolerates changes to infrastructure a easier if modified outside of the state file.
- Permits cross-referencing of resources across state files

COMMERCIAL SUPPORT OPTIONS

- AWS provides CloudFormation support because it is the “House Brand”
- Other open source solutions built on top of CloudFormation are sometimes used, and recommended by, AWS. (eg Troposphere)
- Enhancements, third party tools, etc. exist for Terraform. (eg. Terragrunt).
- Documentation is clearer for CloudFormation. Terraform tends to require cross referencing the AWS CLI, or AWS documentation.

PLANNING, ROLLBACKS, AND RECOVERY

- CloudFormation automatically rolls back changes in the case of a failure.
- In some cases a rollback doesn't take effect, requiring manual intervention. In some cases, support from AWS is required.

- Terraform has no notion of rollback and recovery must be done manually.
- Before applying, Terraform generates a plan. Optionally the plan can be made into a file which can be executed later.
- CloudFormation produces no plan. Essentially you get what you get.

CUSTOMIZING DEPLOYMENTS

- Terraform runs on the local machine, and can include bits of arbitrary code executed during deployments.
  - Arbitrary command, usually shell script
  - Accepts parameters that can be derived from variables in the Terraform script
- CloudFormation can invoke Lambda functions for a similar effect.
  - Run server-side to create custom resources
  - Tricky to develop and debug and require additional setup

SUPPORT FOR FEATURES

- Terraform support for some features can lag behind AWS releases
- CloudFormation supports new features immediately on launch
- Third party Terraform modules do exist, but have their own sets of issues.
  - When incorporated into Terraform mainline, they may cause breaking changes.
  - Sometimes re-rolling infrastructure is necessary.

SELECTING WHAT'S BEST FOR YOU

- Looking to retrofit infrastructure? Or rolling out new infrastructure?
- What is importance of readily available commercial support? Or more self-sufficient?
- Ability to automatically roll-back changes? Or is manual management tolerable?
- Do you want to be a able to review execution plans before attempting to modify architecture? Or do you trust rollbacks?
- Does your deployment process require lots of custom scripts here and there to make things work? Or is it more hands-off?
- Is the availability of the latest and greatest AWS features a priority? Or are you willing to wait a little bit before rolling them out?

CONTACT INFO

Patrick Twohig

Namazu Studios LLC

patrick@namazustudios.com

619.752.4863