GAM401

Riot Games: Standardizing Application Deployments Using Amazon ECS and Terraform

Adam Rozumek

November 28, 2016

amazon web services

RIOT GAMES | GAM 401

Standardizing Application Deployments Using Amazon ECS and Terraform

INTRODUCTIONS

WHO AM I?

ADAM ROZUMEK

SYSTEMS ENGINEER

WHAT TO EXPECT

AMAZON ECS. TERRAFORM. VIDEO GAMES.

1. ECS CONSOLIDATION WINS & LESSONS
   Adapting existing deployments & infrastructure
2. TERRAFORM ENCAPSULATION STRATS
   Object Oriented Development Operations
3. MULTI-GAME MODULAR SERVICE DESIGN
   A different kind of scaling

WHAT TO EXPECT

AMAZON ECS. TERRAFORM. VIDEO GAMES.

1. ECS CONSOLIDATION WINS & LESSONS
   Adapting existing deployments & infrastructure
2. TERRAFORM ENCAPSULATION STRATS
   Object Oriented Development Operations
3. MULTI-GAME MODULAR SERVICE DESIGN
   A different kind of scaling

WHAT TO EXPECT

AMAZON ECS. TERRAFORM. VIDEO GAMES.

1. ECS CONSOLIDATION WINS & LESSONS
   Adapting existing deployments & infrastructure
2. TERRAFORM ENCAPSULATION STRATS
   Object Oriented Development Operations
3. MULTI-GAME MODULAR SERVICE DESIGN
   A different kind of scaling

WHAT TO EXPECT

AMAZON ECS. TERRAFORM. VIDEO GAMES.

1. ECS CONSOLIDATION WINS & LESSONS
   Adapting existing deployments & infrastructure
2. TERRAFORM ENCAPSULATION STRATS
   Object Oriented Development Operations
3. MULTI-GAME MODULAR SERVICE DESIGN
   A different kind of scaling


LEAGUE
LEGENDS


SK TELECOM T1
SAMSUNG GALAXY
DELTA
verizon

2016
LEAGUE OF LEGENDS STATS

MORE THAN
100MILLION

MONTHLY ACTIVE
PLAYERS

MORE THAN
27MILLION

DAILY ACTIVE
PLAYERS

MORE THAN
7.5MILLION

PEAK CONCURRENT
PLAYERS

DATA PRODUCTS & SERVICES

OUR MISSION

Empower teams at Riot to make timely, data-informed products by maintaining a scalable and reliable data platform

PROBLEM

PROBLEM

SCALING TOTAL OWNERSHIP ON AWS

Total ownership

We want to **empower** developers to:

- Provision their own infrastructure
- Execute their own deployments
- Monitor their own metrics

PROBLEM

SCALING TOTAL OWNERSHIP ON AWS

Resource attribution

- Who owns these EBS volumes?
- What applications depend on these security groups?
- Can these AMIs be deleted?

PROBLEM

SCALING TOTAL OWNERSHIP ON AWS

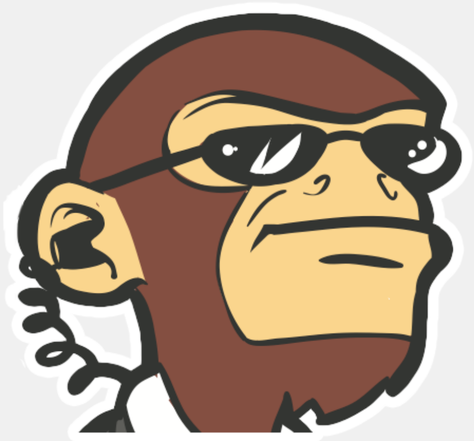
Security

Custom TCP Rule	TCP	50000	sg-2a598753 (vertica_bogdan_vertica_internal_traffic)
Custom TCP Rule	TCP	5434	sg-2a598753 (vertica_bogdan_vertica_internal_traffic)
Custom ICMP Rule	Traceroute	N/A	sg-2a598753 (vertica_bogdan_vertica_internal_traffic)
DNS (UDP)	UDP	53	10.93.0.0/22
Custom TCP Rule	TCP	4803 - 4805	sg-2a598753 (vertica_bogdan_vertica_internal_traffic)
Custom TCP Rule	TCP	5444	sg-2a598753 (vertica_bogdan_vertica_internal_traffic)
Custom ICMP Rule	Echo Reply	N/A	sg-2a598753 (vertica_bogdan_vertica_internal_traffic)
All traffic	All	All	0.0.0.0/0

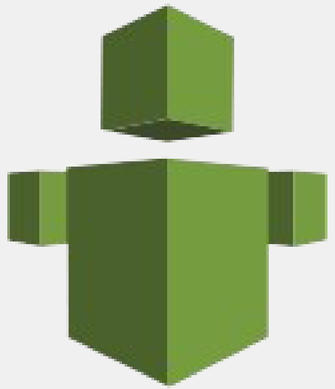
PROBLEM
SCALING TOTAL OWNERSHIP ON AWS

Security

- Auditing is important, but it’s reactive
- Operational time sink



Security Monkey



AWS Trusted Advisor

PROBLEM
SCALING TOTAL OWNERSHIP ON AWS

Enforcing conventions

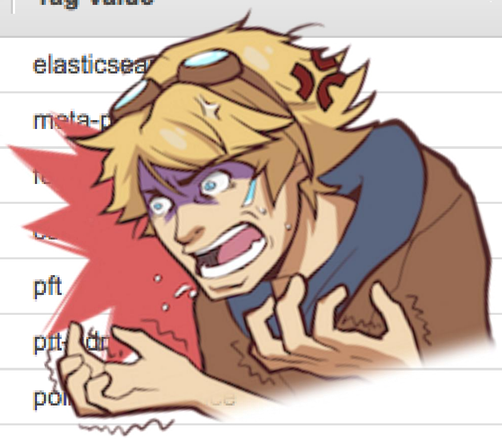
- Common syntax for tags
- Simplify reservations
- Standardize networks

PROBLEM
SCALING TOTAL OWNERSHIP ON AWS

Enforcing conventions

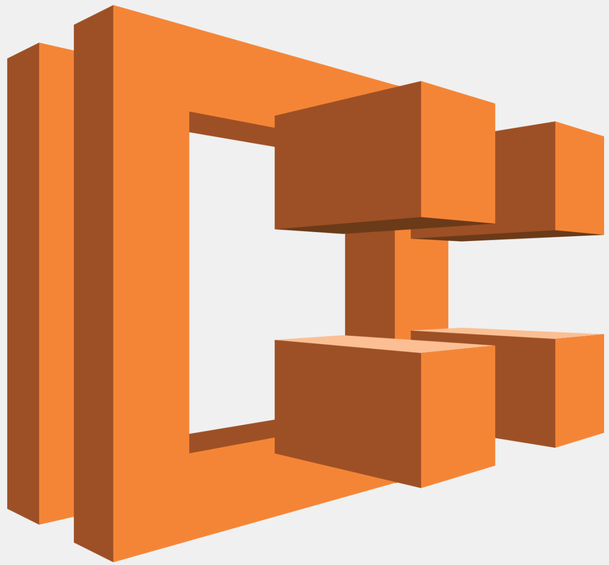
- Common syntax for tags
- Simplify reservations
- Standardize networks

	Tag Key	Tag Value	Total
Manage Tag	App	elasticsea	33
Manage Tag	App	meta-p	10
Manage Tag	app		1
Manage Tag	app		1
Manage Tag	App_Name	pft	4
Manage Tag	App_Name	pft-[illegible]	4
Manage Tag	application	po[illegible]	51
Manage Tag	AppName	legs-prod-proxy-west2	9



PROBLEM

SCALING TOTAL OWNERSHIP ON AWS



Amazon EC2
Container Service

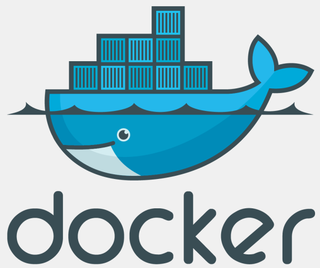
Terraform

CONTAINERS

CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

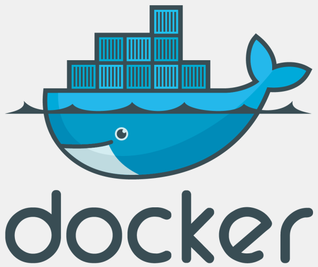
- Dockerfiles capture application dependencies



CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

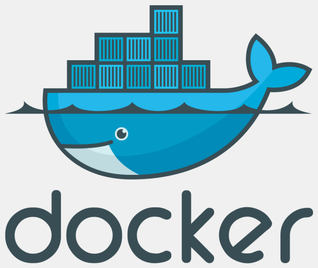
- Dockerfiles capture application dependencies
- Common use cases have great community support



CONTAINERS

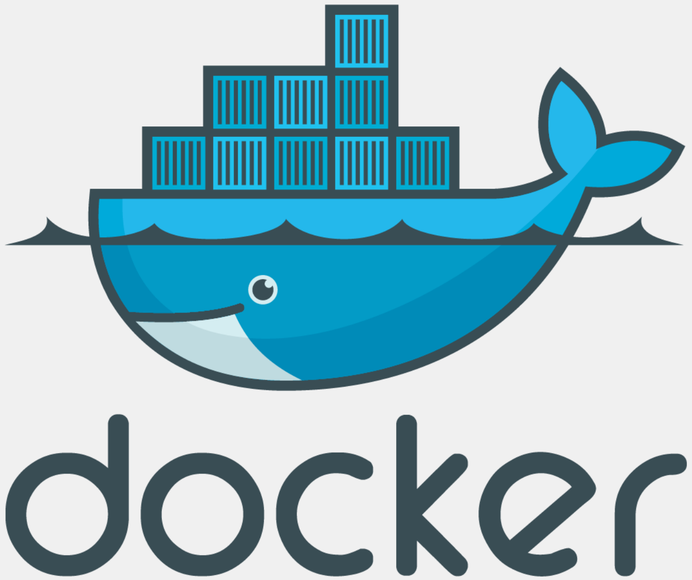
STANDARDIZED APPLICATION UNITS… ON AWS

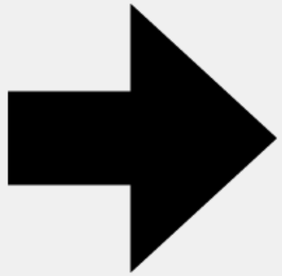
- Dockerfiles capture application dependencies
- Common use cases have great community support
- Profit from our own engineering community



CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS





CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

Embrace the abstraction

- Plan for failure at all levels
- Avoid manual intervention whenever possible
- It’s ephemeral all the way down

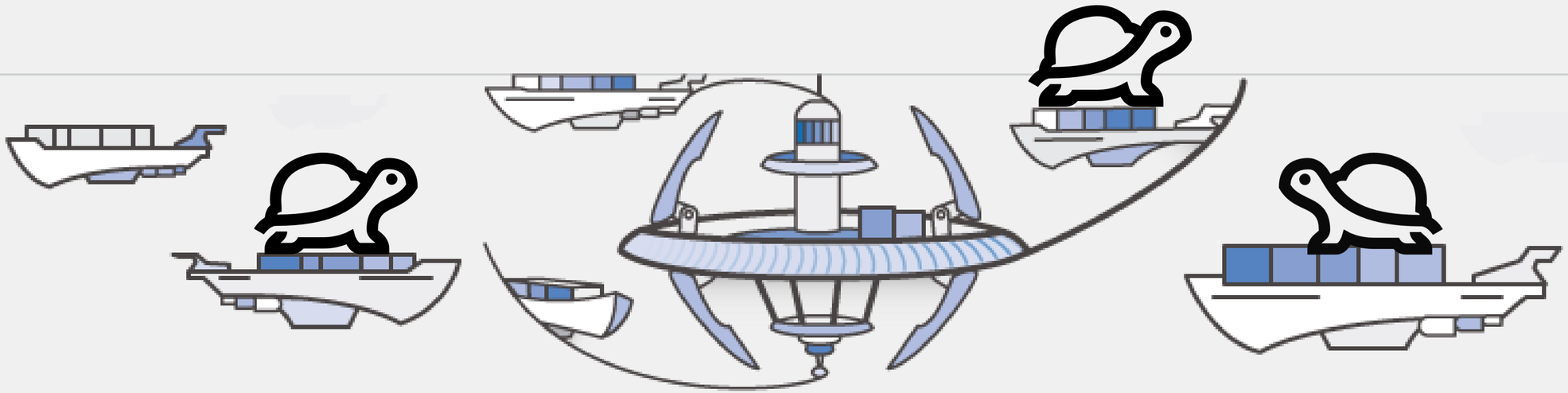
CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

Scheduling is hard

We need to:

- Quickly and “fairly” run tasks
- Prevent resource conflicts
- Provide reasonable fault tolerance

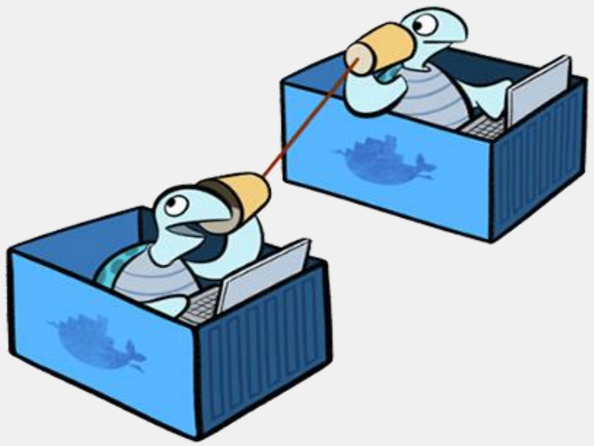


CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

We need AWS hardware

- Enable total ownership of the AWS resources backing our containers
- Avoid the security, resource attribution, and convention degradation pitfalls



HISTORY

CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

First iteration (2013)

- Traditional ASGs
- AMIs configured to launch a single Docker container
- Managed by Netflix’s Asgard

CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

First iteration (2013)

- Easy to adopt
- Familiar AWS concepts

CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

First iteration (2013)

- No increased resource utilization
- No elasticity improvements

CONTAINERS

STANDARDIZED APPLICATION UNITS… ON AWS

Second iteration (2013)

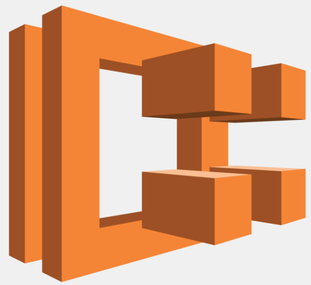
- Third party container orchestration
- Production scale had unexpected challenges

CONTAINERS

STANDARDIZED APPLICATION UNITS… ON ECS!

Third iteration: Amazon EC2 Container Service

- ECS AMI provides all necessary software
- Designed with AWS integration in mind
- Free!

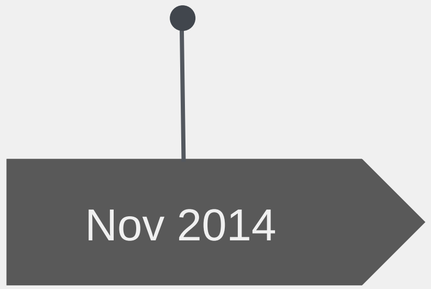


AMAZON EC2 CONTAINER SERVICE

KEY FEATURE TIMELINE

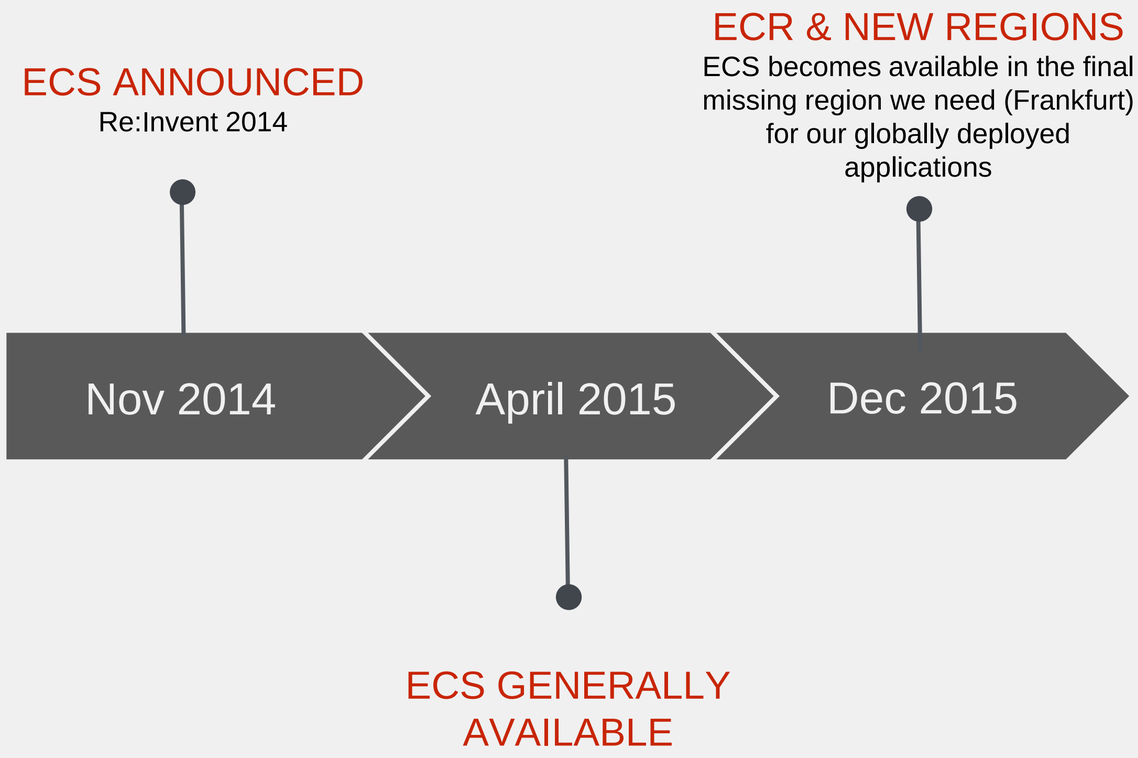
ECS ANNOUNCED

Re:Invent 2014



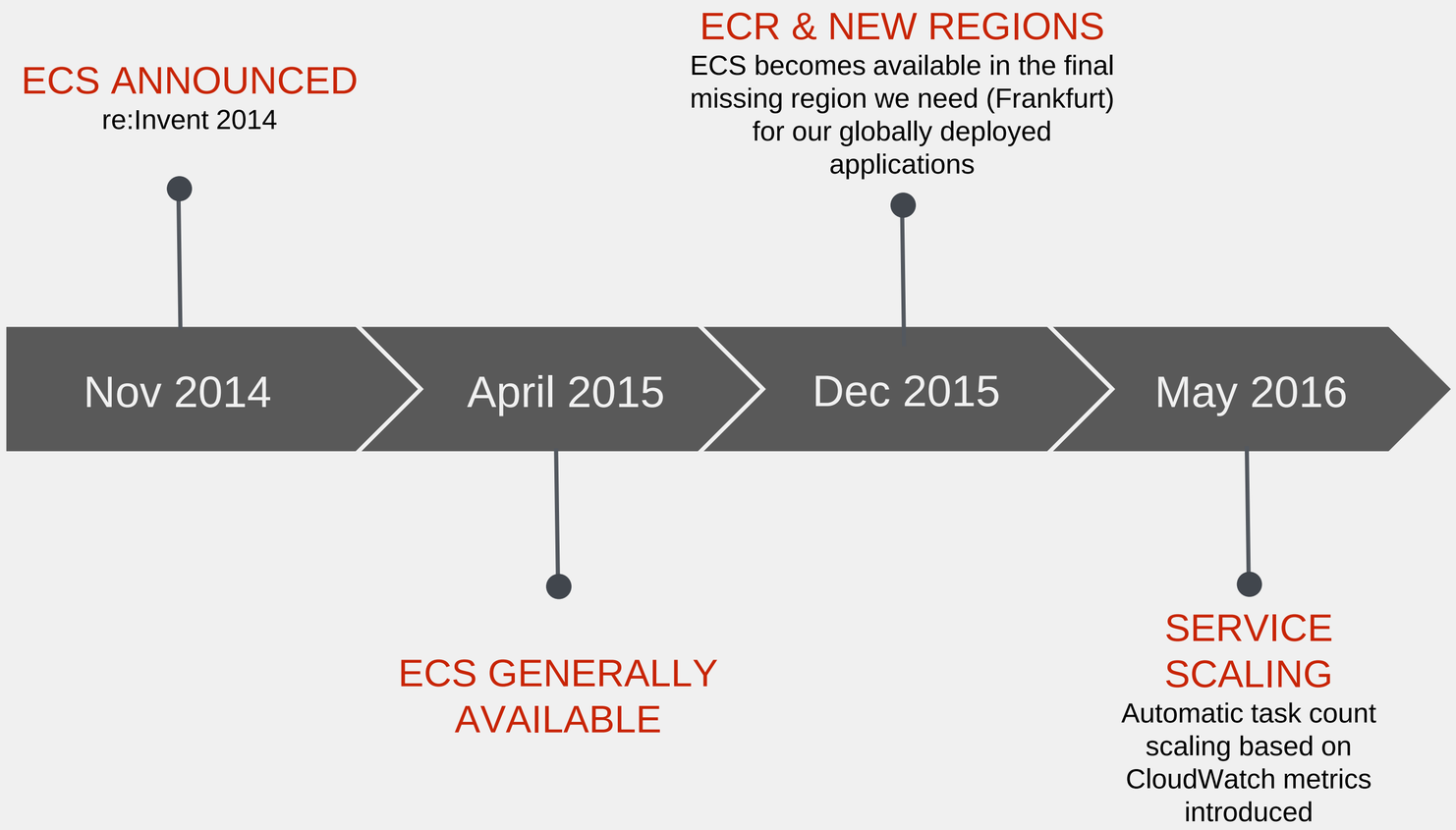
AMAZON EC2 CONTAINER SERVICE

KEY FEATURE TIMELINE



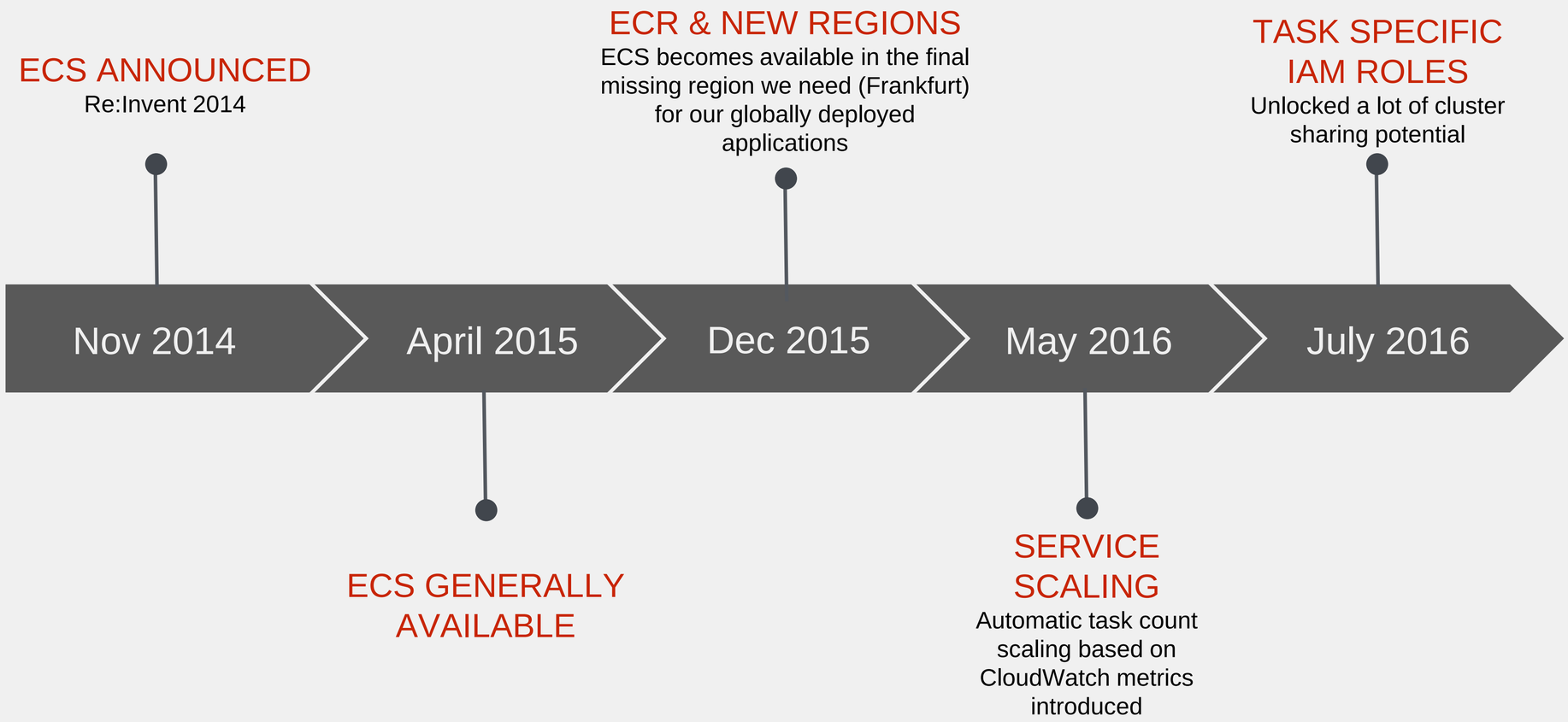
AMAZON EC2 CONTAINER SERVICE

KEY FEATURE TIMELINE



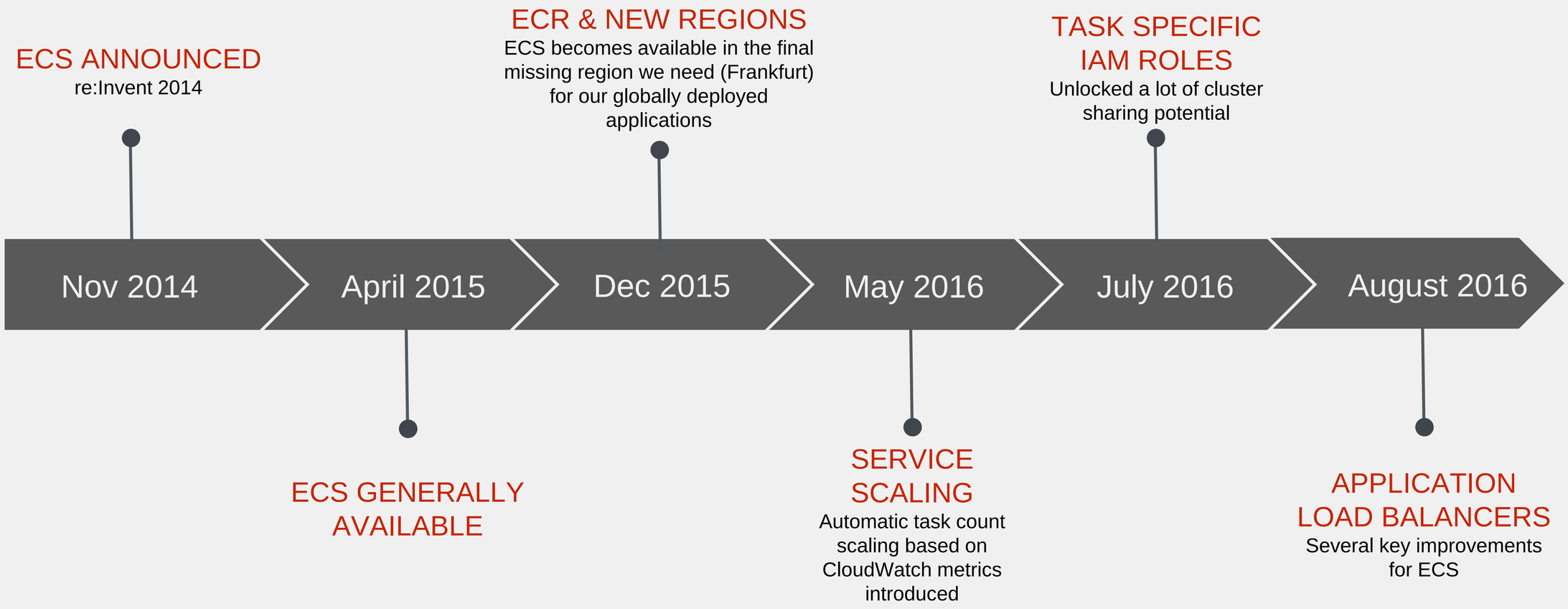
AMAZON EC2 CONTAINER SERVICE

KEY FEATURE TIMELINE



AMAZON EC2 CONTAINER SERVICE

KEY FEATURE TIMELINE



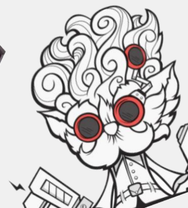
BEYOND ECS
INFRASTRUCTURE AS CODE

- At scale, orchestrating infrastructure in a consistent, reproducible way is key

BEYOND ECS
INFRASTRUCTURE AS CODE

- At scale, orchestrating infrastructure in a consistent, reproducible way is key

Total ownership



TERRAFORM

TERRAFORM

INFRASTRUCTURE AS OBJECT-ORIENTED CODE

- Intelligent & flexible flow control

TERRAFORM

INFRASTRUCTURE AS OBJECT-ORIENTED CODE

- Intelligent & flexible flow control
- Configuration drift managed

TERRAFORM

INFRASTRUCTURE AS OBJECT-ORIENTED CODE

- Intelligent & flexible flow control
- Configuration drift managed
- AWS resources documented

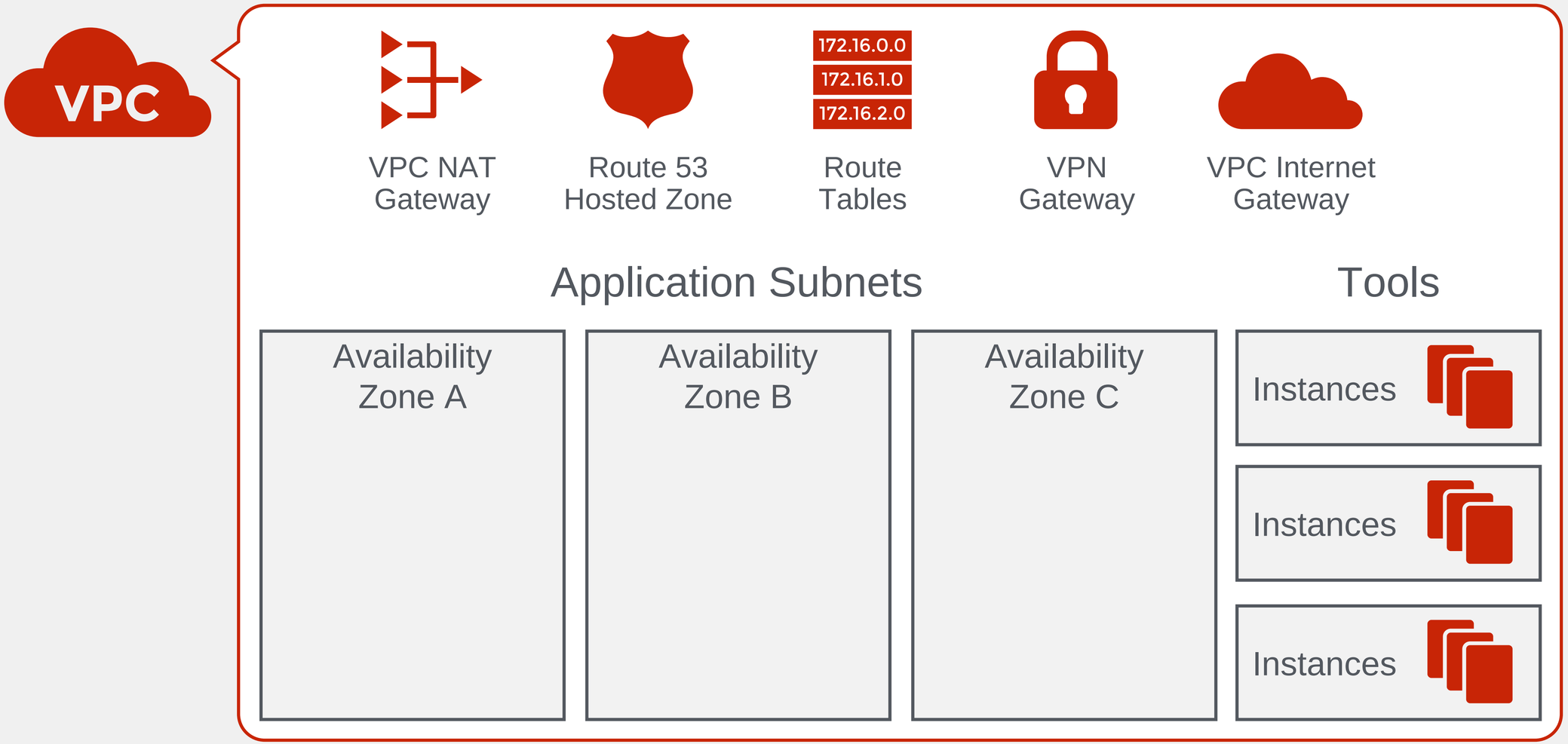
TERRAFORM

INFRASTRUCTURE AS OBJECT-ORIENTED CODE

- Intelligent & flexible flow control
- Configuration drift managed
- AWS resources documented
- Share common use cases

TERRAFORM

INFRASTRUCTURE AS OBJECT-ORIENTED CODE



TERRAFORM

INFRASTRUCTURE AS OBJECT-ORIENTED CODE

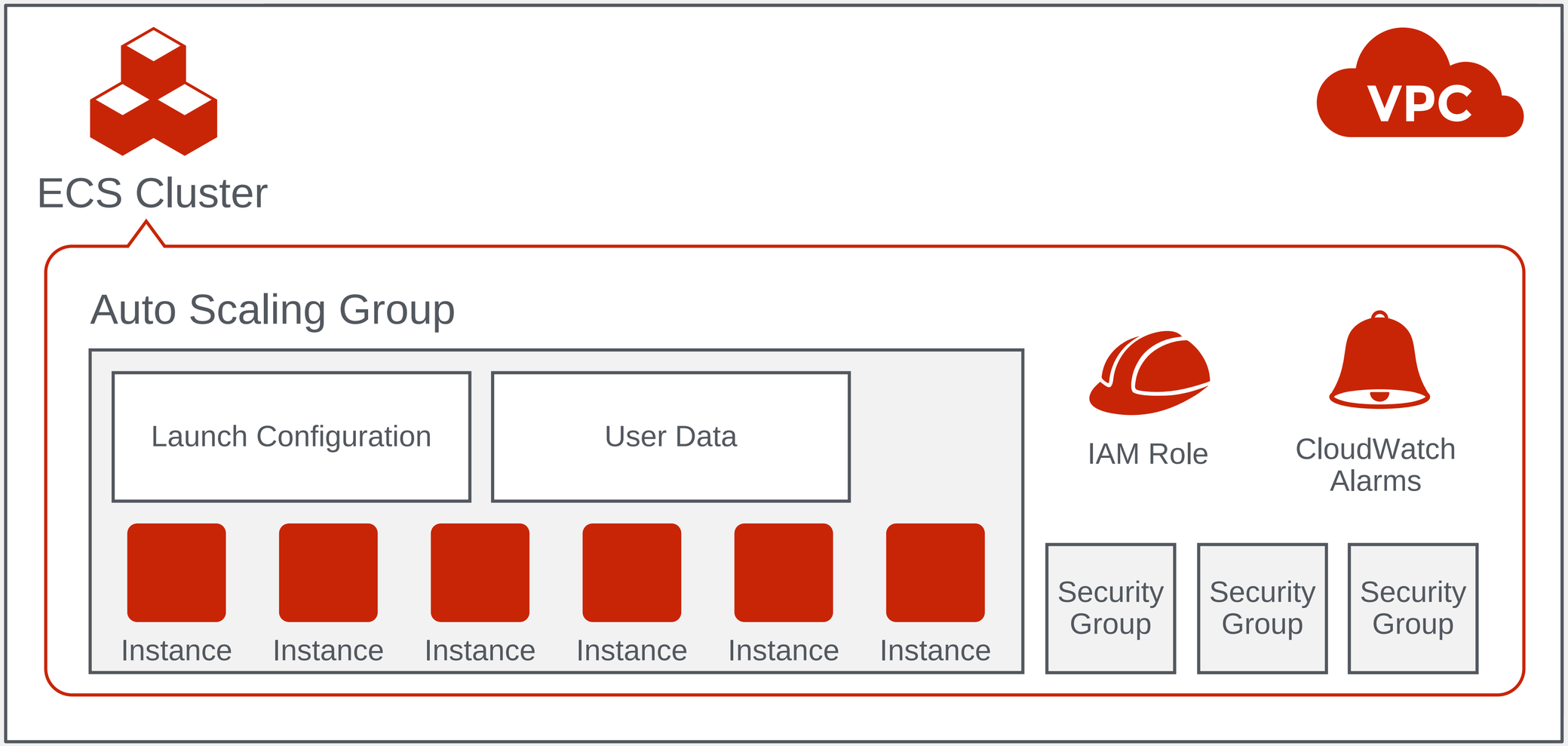
Simplified infrastructure definitions

```
# Your AWS account ID
account_id = "012345678901"

# Configure the AWS region you are in.
region = "us-east-1"

# Set the name of your INITATIVE here
account_name = "rpg"

# This is the name of your VPC
vpc_name = "sandbox_vpc"

# Google Group Distribution for your team
owner = "myteamalias@riotgames.com"

# REQUIRED resource attribution tag
accounting = "territory.initiative.project"

# You should update this to the subnet that is allotted to you
vpc_cidr = "10.181.180.0/22"
azs = "us-east-1a,us-east-1c"

# USER CONFIGURED. This is an example of subnet distribution
frontend_subnets = "10.180.161.0/27,10.180.161.32/27"
backend_subnets = "10.180.161.128/27,10.180.161.160/27"
frontend_dev_subnets = "10.180.162.0/27,10.180.162.32/27"
backend_dev_subnets = "10.180.162.128/27,10.180.162.160/27"
```

CASE STUDIES

ECS CLUSTER

TERRAFORM BUILDING BLOCKS



ECS CLUSTER

TERRAFORM BUILDING BLOCKS

Module 1

```
output "private_subnets" {
  value = "${join(",", aws_subnet.private.*.id)}"
}
```

```
output "team_access_security_group" {
  value = "${aws_security_group.team_access.id}"
}
```

Module 2

Module 3

```
module "ecs-autoscaling" {

    source = "./ecs-autoscaling"
    cluster_name = "reinvent-demo-large-static-cluster"

    # Tag values
    owner = "arozumek"
    team = "dps@riotgames.com"

    # VPC Information
    availability_zones = "${module.vpc.availability_zones}"
    subnet_ids = "${module.vpc.private_subnet_ids}"

    # Instance configuration
    key_name = "reinvent-keypair"
    instance_type = "r3.4xlarge"
    security_group_ids="${var.team_access_sg}"
    iam_instance_profile = "Generic-ECS"

    # ASG configuration
    min_size = "3"
    max_size = "3"
    desired_capacity ="3"

    # Docker registry
    registry_url = "registry.rcluster.io"
    registry_email = "${var.module_registry_email}"
    registry_auth = "${var.module_registry_auth}"

}
```

ECS CLUSTER

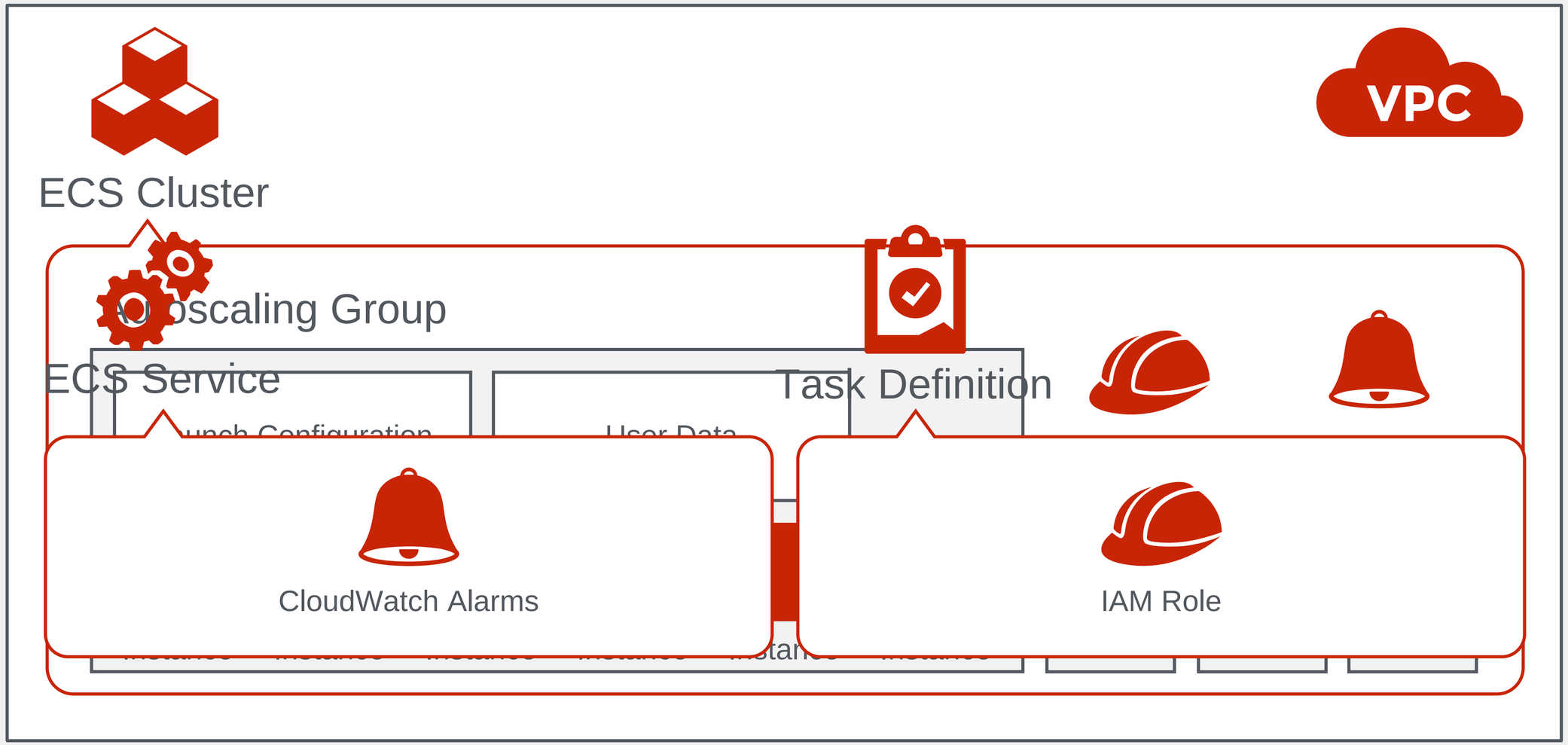
TERRAFORM BUILDING BLOCKS

```
module "ecs-autoscaling" {

    source = "./ecs-autoscaling"
    cluster_name = "reinvent-demo"

    # Tag values
    owner = "arozumek"
    team = "dps@riotgames.com"

    # VPC Information
    availability_zones = "${module.vpc.availability_zones}"
    subnet_ids = "${module.vpc.private_subnet_ids}"

    # Instance configuration
    key_name = "reinvent-keypair"
    instance_type = "m3.large"
    security_group_ids="sg-d7b890b3"
    iam_instance_profile = "Generic-ECS"

    # ASG configuration
    min_size = "3"
    max_size = "10"
    desired_capacity ="4"

    # Docker registry
    registry_url = "registry.rcluster.io"
    registry_email = "${var.module_registry_email}"
    registry_auth = "${var.module_registry_auth}"

}
```

```
module "ecs-autoscaling" {

    source = "./ecs-autoscaling"
    cluster_name = "reinvent-demo"

    # Tag values
    owner = "arozumek"
    team = "dps@riotgames.com"

    # VPC Information
    availability_zones = "${module.vpc.availability_zones}"
    subnet_ids = "${module.vpc.private_subnet_ids}"

    # Instance configuration
    key_name = "reinvent-keypair"
    instance_type = "r3.4xlarge"
    security_group_ids="sg-d7b890b3"
    iam_instance_profile = "Generic-ECS"

    # ASG configuration
    min_size = "3"
    max_size = "3"
    desired_capacity ="3"

    # Docker registry
    registry_url = "registry.rcluster.io"
    registry_email = "${var.module_registry_email}"
    registry_auth = "${var.module_registry_auth}"

}
```

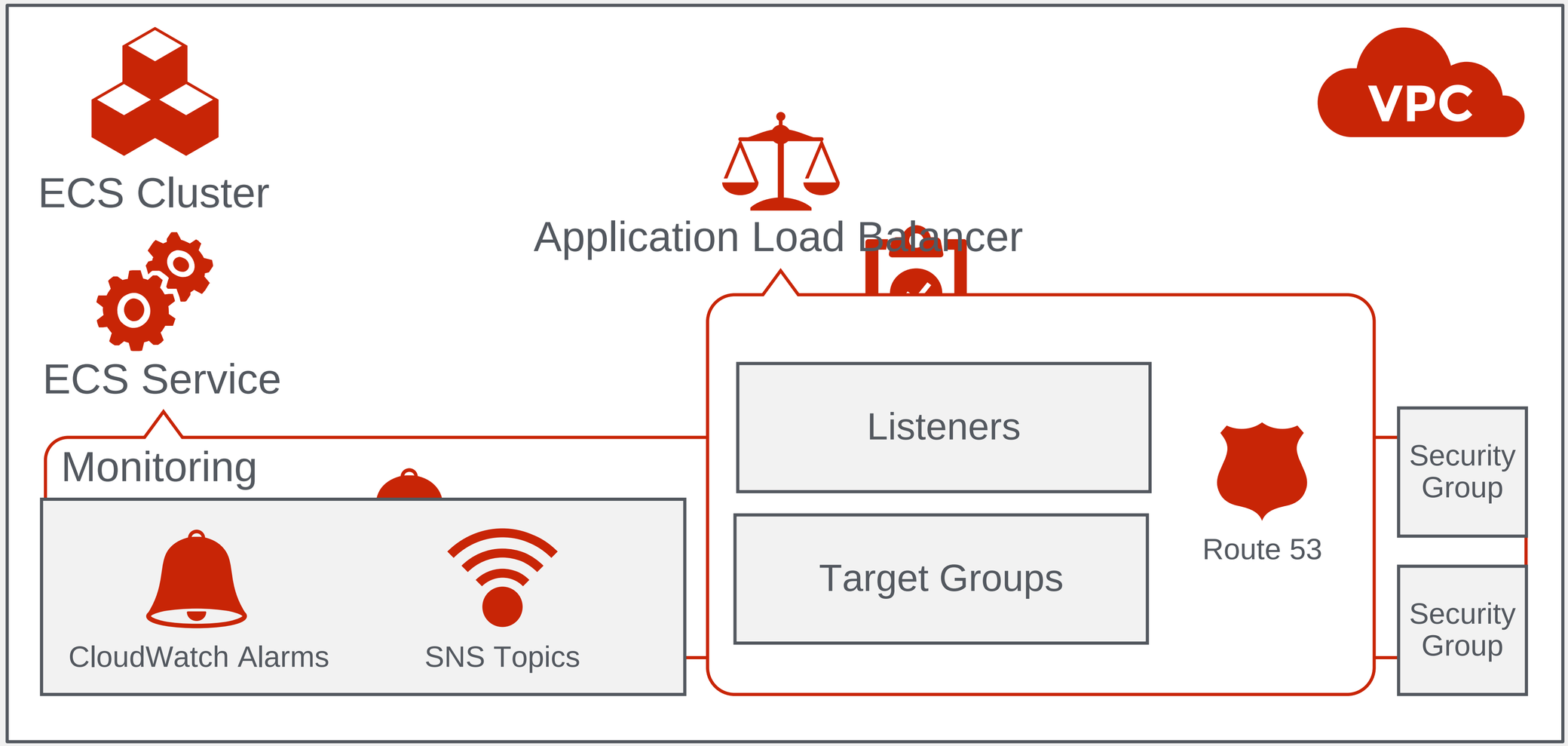
MICROSERVICES

WITHOUT SERVICE ENDPOINTS



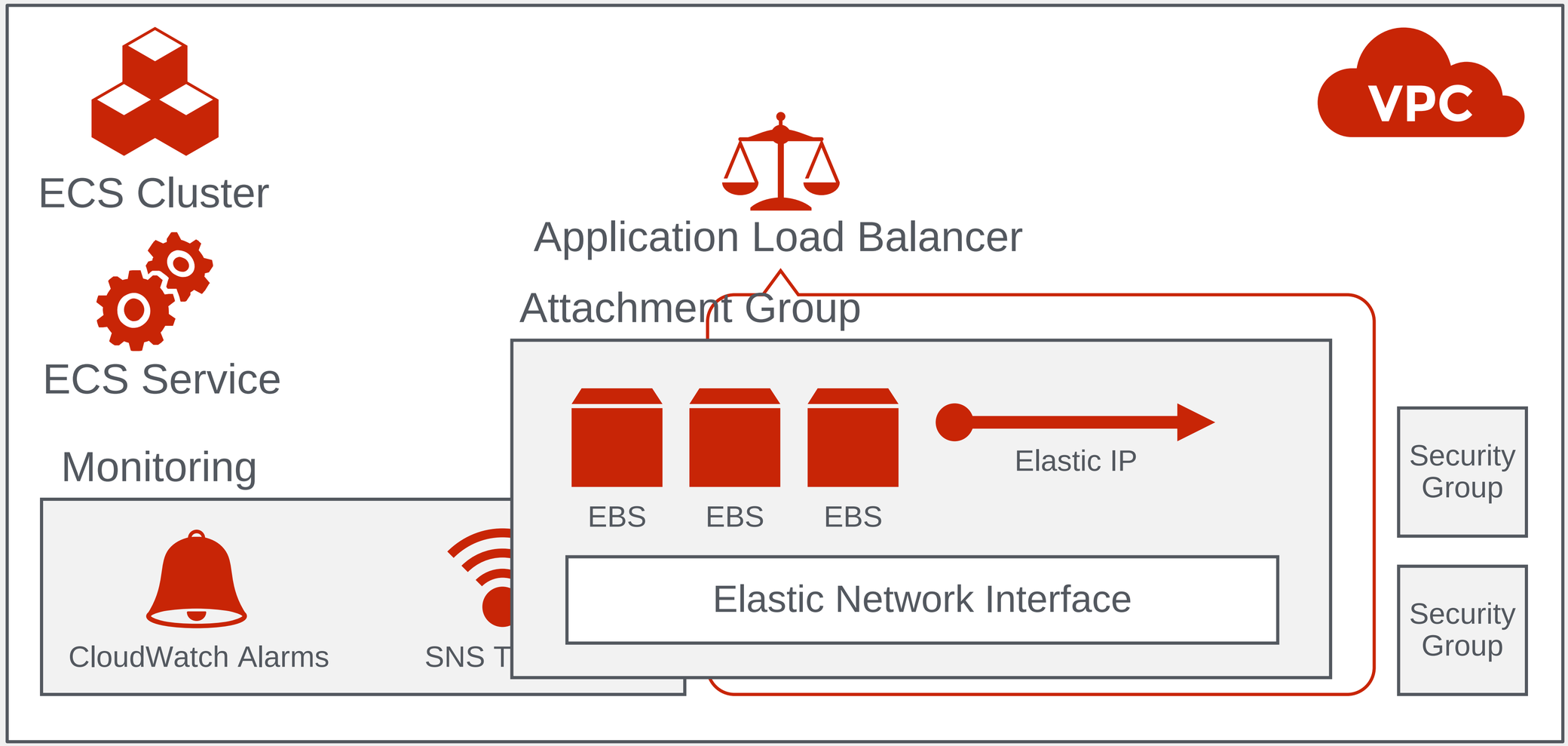
MICROSERVICES

WITH SERVICE ENDPOINTS



PERSISTENT DATA

LOSE ECS HOSTS WITHOUT LOSING DATA



LESSONS LEARNED

LESSONS

WHAT WORKED FOR US

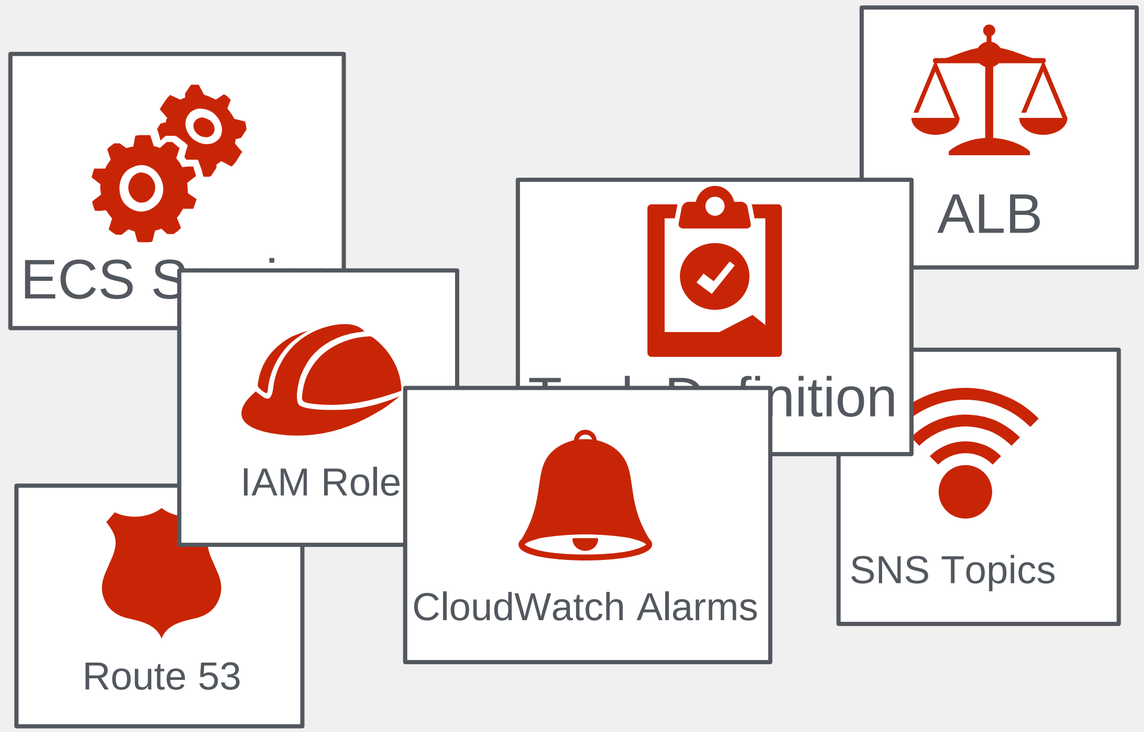
- Socialize your templates & take advantage of the community!
- Terraform community modules
- Terraform AWS blog

LESSONS
WHAT WORKED FOR US

- Break apart your stacks

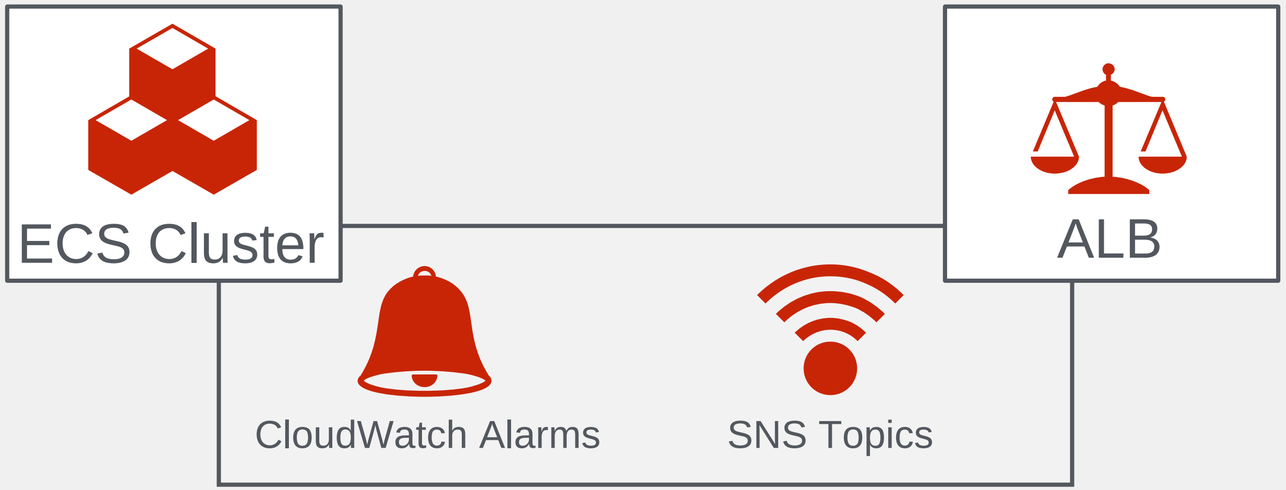
LESSONS
WHAT WORKED FOR US

- Break apart your stacks, but don’t overdo it



LESSONS
WHAT WORKED FOR US

- Break apart your stacks, but don’t overdo it
- Use modules to reduce code duplication



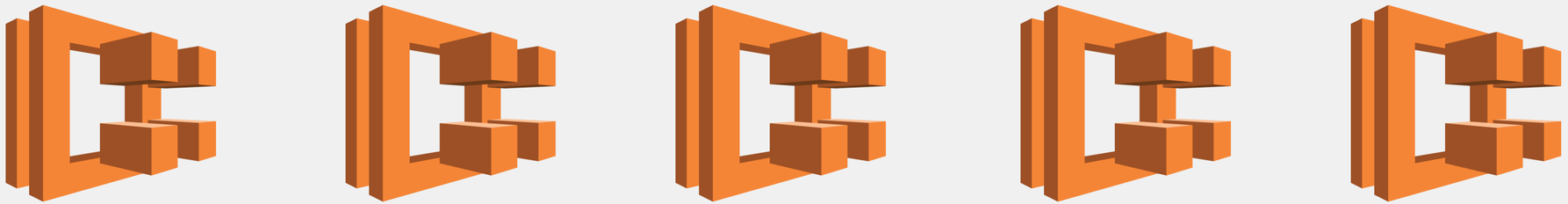
LESSONS
WHAT WORKED FOR US

- Use remote state files

```
data "terraform_remote_state" "foo" {
    backend = "s3"
    config {
        bucket = "terraform-state-prod"
        key = "network/terraform.tfstate"
        region = "us-east-1"
    }
}
```

LESSONS
WHAT WORKED FOR US

- Be liberal with your cluster provisioning
- Don’t risk resource contention in production
- With good orchestration, additional clusters != additional operational overhead



LESSONS
WHAT WORKED FOR US

- Tag everything all of the time
- Keep your tags organized in your Terraform templates
- Have top level variables that get applied to every resource
- Create a tag for every dimension that is useful to your business

LESSONS
WHAT WORKED FOR US

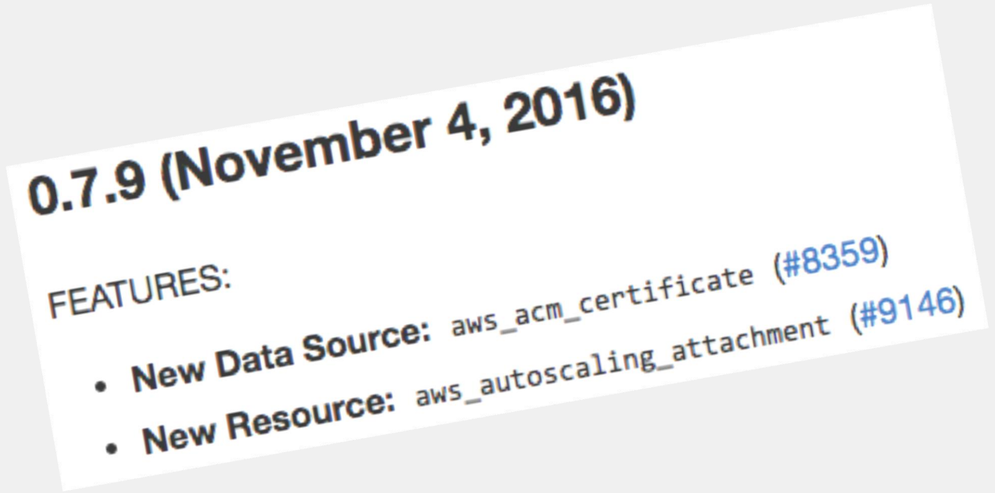
- Centralize your logs

or

Amazon
CloudWatch
Logs

LESSONS
WHAT WORKED FOR US

- Stay up to date with release blogs and application updates
- ECS updates on the AWS blog
- Terraform’s open source GitHub repository changelog



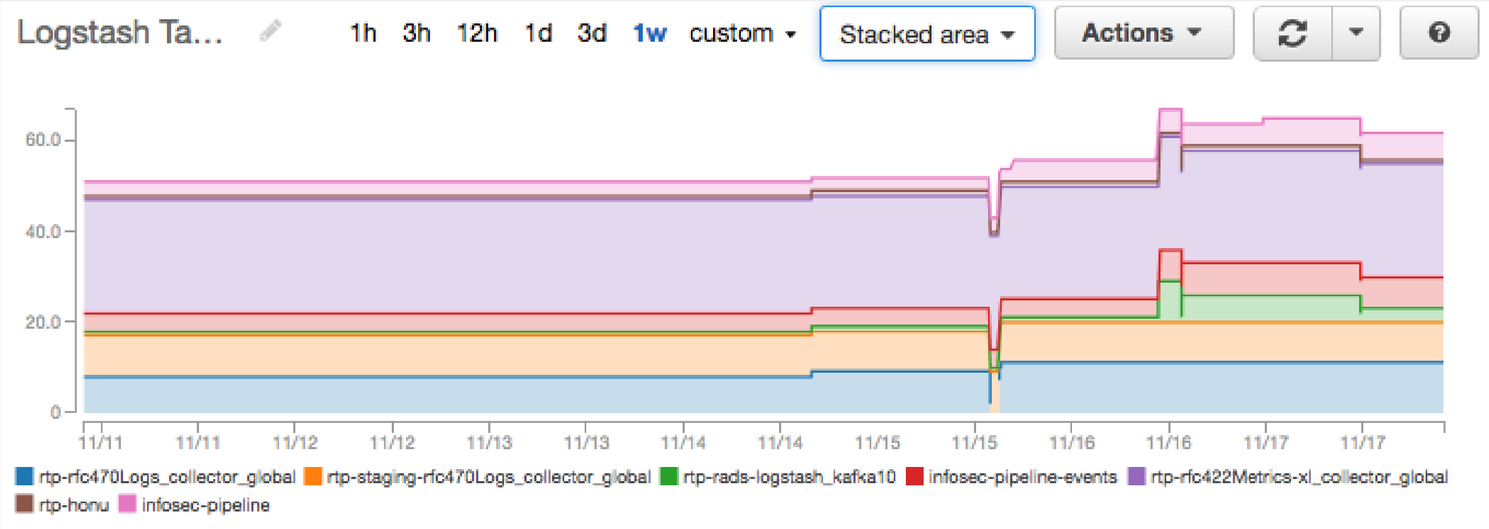
LESSONS
WHAT WORKED FOR US

- Capture your AMI generation process

```yaml
- hosts: "{{ env }}"
  remote_user: ec2-user
  become: yes
  become_user: root


  vars:
    logstash_version: 1.5.5
    shield_logstash_username: logstash

  tasks:

  - name: Include Vault secured variables
    include_vars: ../conf/es.vault.yml

  # Install / Configure NTP
  - name: Install / Configure NTP
    include: ../includes/support_modules/ntp_setup.yaml

  # Perform Logstash Setup Operations
  - name: Perform Logstash Setup
    include: ../includes/logstash_setup.yaml
```

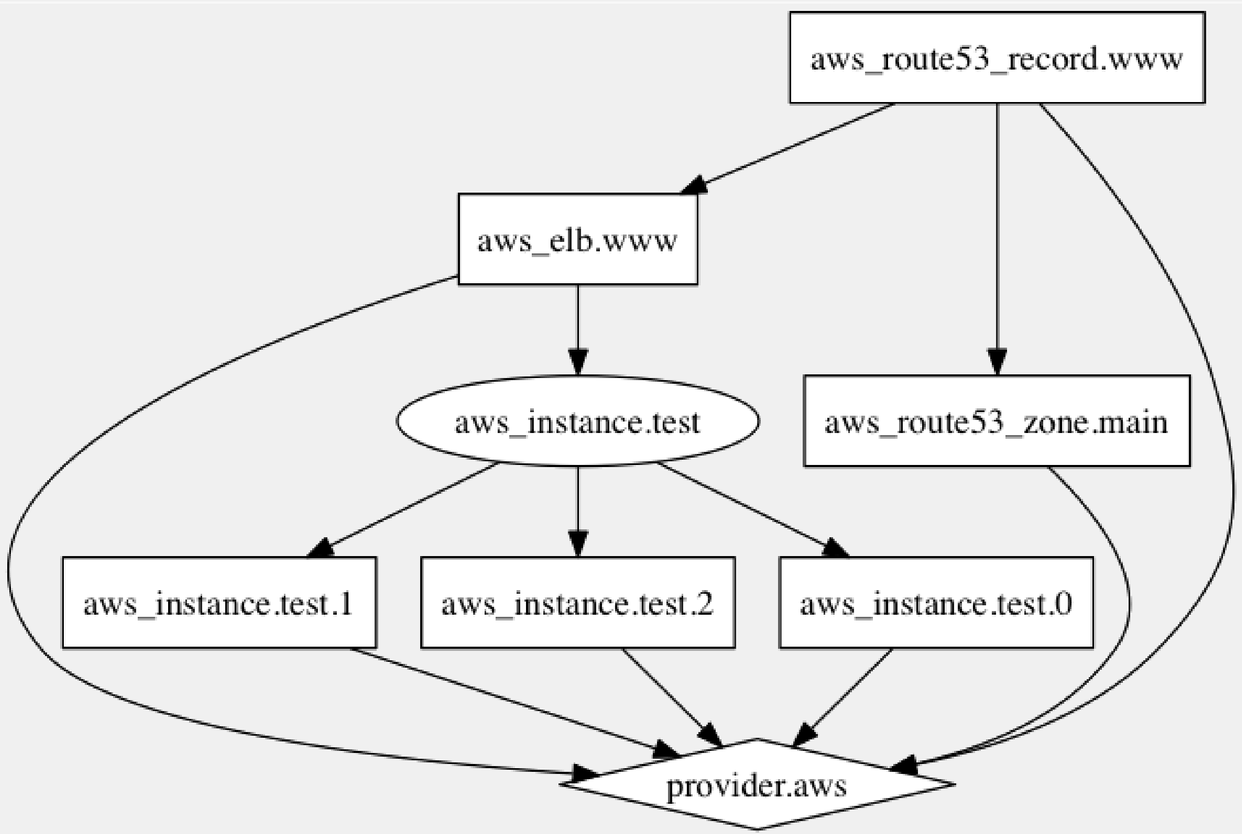
LESSONS
WHAT WORKED FOR US

- Profile your memory requirements, monitor for scheduling issues



LESSONS
WHAT WORKED FOR US

- Take advantage of flow control in Terraform
- Use the Terraform graph tool to visualize your dependencies



RIOT ENGINEERING BLOG

THESE PROBLEMS AND MORE

Riot engineering

https://engineering.riotgames.com/

How we use data

http://na.leagueoflegends.com/en/tag/insights

Tuesday
November 29th, 2016
6:30 – 10:30PM

Paiza Lounge
The Venetian
36th Floor

ENJOY NOMS.

DRINK DRINKS.

PLAY GAMES.

JOIN US!

Thank you!

Remember to complete your evaluations!