Immutable Servers

Building a deployment pipeline and deploying to EC2 Spot Instances

@james_ridgway

Who am I?

- My name is James Ridgway
- I work on the Dev side of DevOps
- Head of Platform and Data Science at The Floow
  - Building services to run on our infrastructure
  - Looking at the insights we can generate from the insights our data shows
- I'm also a typical nerd... I like to tinker with stuff...

Overview

- The Immutable Servers Pattern
- An Introduction to Spot Instances
- Building a pipeline
    - AMIs with Packer
    - Infrastructure with Terraform
    - CI/CD with Jenkins

The Immutable Servers Pattern

"A server that once deployed, is never modified, only ever replaced".

Why Immutable Servers and Spot Instances?

Yelp runs Apache Mesos on AWS Spot Instances.

The Challenge

I thought it would be a fun challenge to:

1. Migrate my personal projects off of my current VPS hosting to AWS
2. Use this as an excuse to play with Spot Instances and adopt an Immutable Servers pattern
3. Try and use AWS... cheaply

Change Management Evolution

- Snowflake Servers
- Configuration Management Tooling
- Phoenix Servers
- Immutable Servers

Snowflake Servers

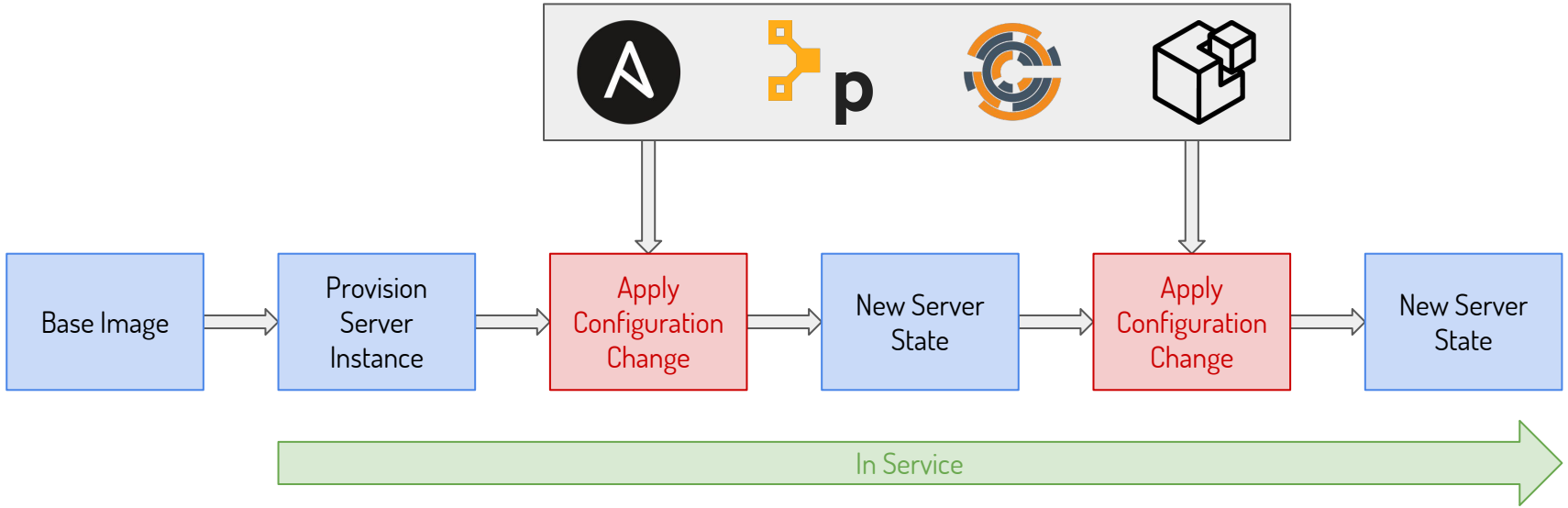
Definition:

- Manually applied:
  - Patches
  - Updates
  - Configuration Tweaks
  - Dependency Updates
  - Code Changes
- No configuration management

The Problems:

- Difficult to reproduce
- Spinning up a new instance or environments is incredibly time consuming
- Development/testing/staging is unlikely to mirror production
- Need manual processes and documentation to form an audit trail

Configuration Management Tooling



Configuration Management Tooling

Advantages:

- Controlled changes via version controlled recipes, manifests, etc.
- Changes can be rapidly applied to servers

Challenges

- Inconsistencies can occur over time
  - Manual, undocumented changes
- The longer the server is provisioned the greater the chance of configuration drift
- Can only manage what it knows about

Phoenix Servers

Regularly rebuild a servers from the base image:

Phoenix Servers

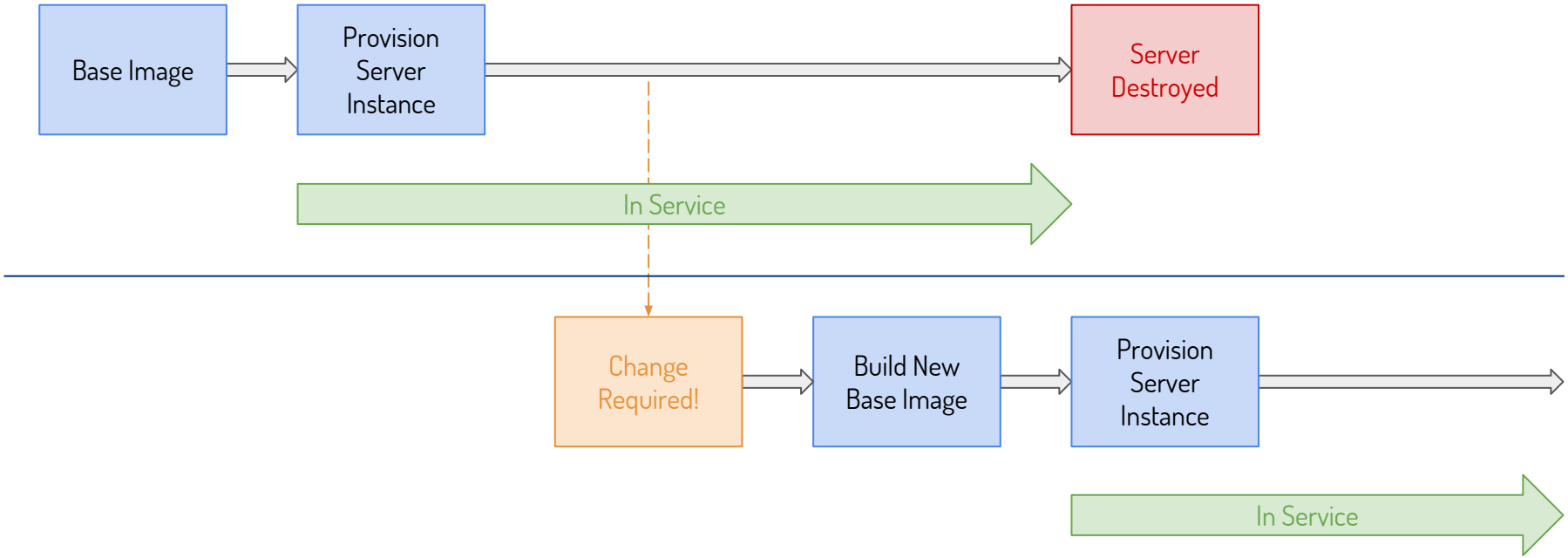
Advantages:

- Controlled changes via version controlled recipes, manifests, etc.
- Changes can be rapidly applied to servers
- Easily replicate changes to a point in time
- Focus shifts towards fixing issues in the base image.

Challenges

- Inconsistencies can occur over time
  - Manual, undocumented changes
- The longer the server is provisioned the greater the chance of configuration drift
- Can only manage what it knows about

Immutable Servers



Immutable Servers

Advantages:

- Fully defined in code (config tooling or scripts)
- Easily replicate changes to a point in time
- Base image typically better suited to horizontal scaling
- Every server built from the same image should be consistent
- Can decrease attack service by preventing remote access.
- Fixed configuration - no config drift

Challenges

- Debugging issues - tooling not built into image
- Applying changes can be slower than alternative patterns (snowflake servers, config management tooling phoenix servers, etc)
- Not all applications are suited (e.g. databases)

Pets vs Cattle

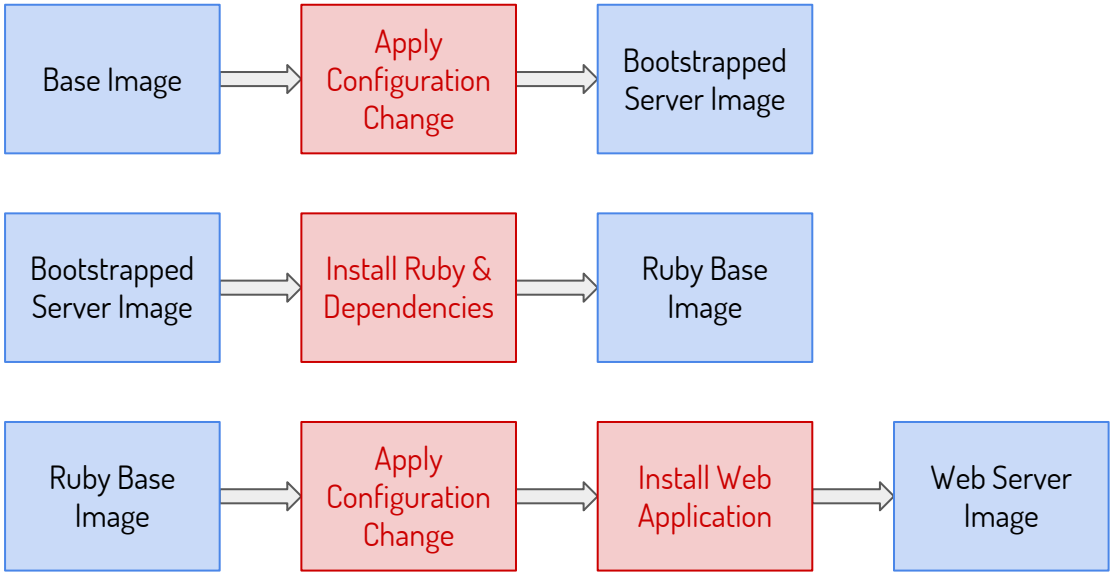
"In the old way of doing things, we treat our servers like pets, for example Bob the mail server. If Bob goes down, it's all hands on deck. The CEO can't get his email and it's the end of the world. In the new way, servers are numbered, like cattle in a herd. For example, www001 to www100. When one server goes down, it's taken out back, shot, and replaced on the line."

Randy Bias

Immutable Servers - The Challenges

- Debugging
  - Bake tooling in over time
- Changes can be slow
  - Allow emergency fixes to be applied manually (purists will insist on no changes)
  - Minimise your risk of configuration drift - e.g. flag the server for deletion after a week if a manual change is detected
- Not all applications are suited (e.g. Databases)
  - Forces you to separate state/data; make your servers stateless
  - Centrally store data (e.g. EBS/EFS/NFS/SAN, etc)
  - Or make it someone else's problem (e.g DBaaS, RDS, etc)

Immutable Servers – Layer Images



Immutable Servers - Requirements

- Automated build pipeline and deployment process
  - Deployment and rollback can be the same process
  - Blue-green deployments, etc
- Backwards compatible data structures
- Push configuration down into base image
  - Doesn't strictly require config management tooling (SaltStack, Puppet, Chef, etc.)
  - Forces loose coupling with infrastructure and other services
  - Instance configuration external to the server
- Clear separation between stateful and ephemeral data

AWS EC2 Spot Instance

AWS EC2

On-demand:

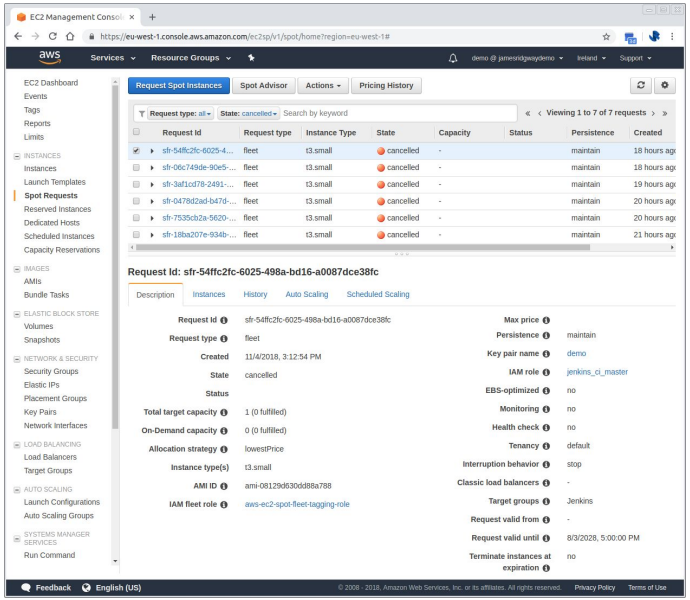
- Pay per hour at the full rate

Reserved Instances:

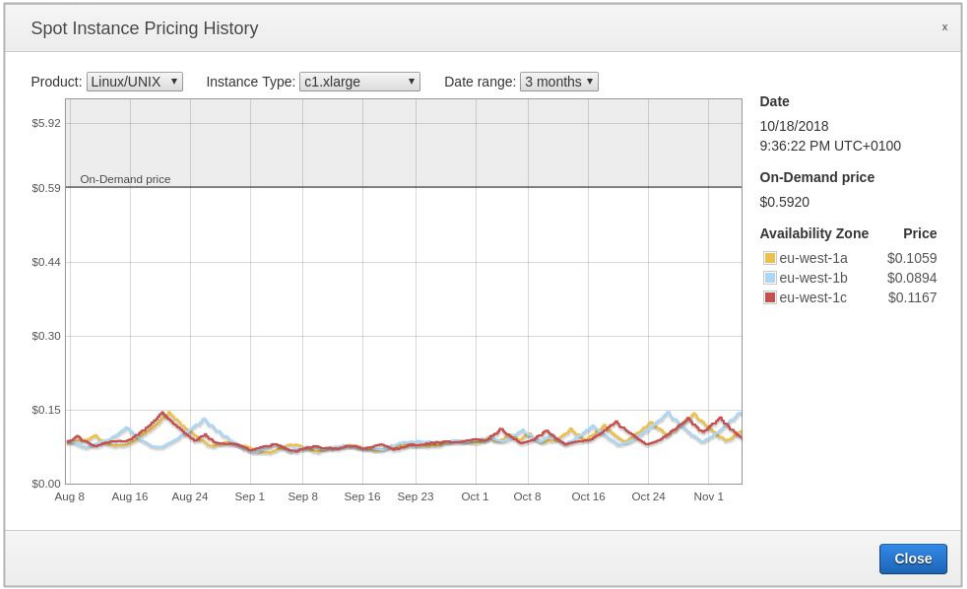
- Standard RI: upto 75% discount on on-demand pricing
  - No upfront, partial upfront or all upfront costings
- Convertible RI: upto 54% off on-demand pricing
  - Can change instance type, instance family, tenancy, etc

AWS EC2 Spot Instance

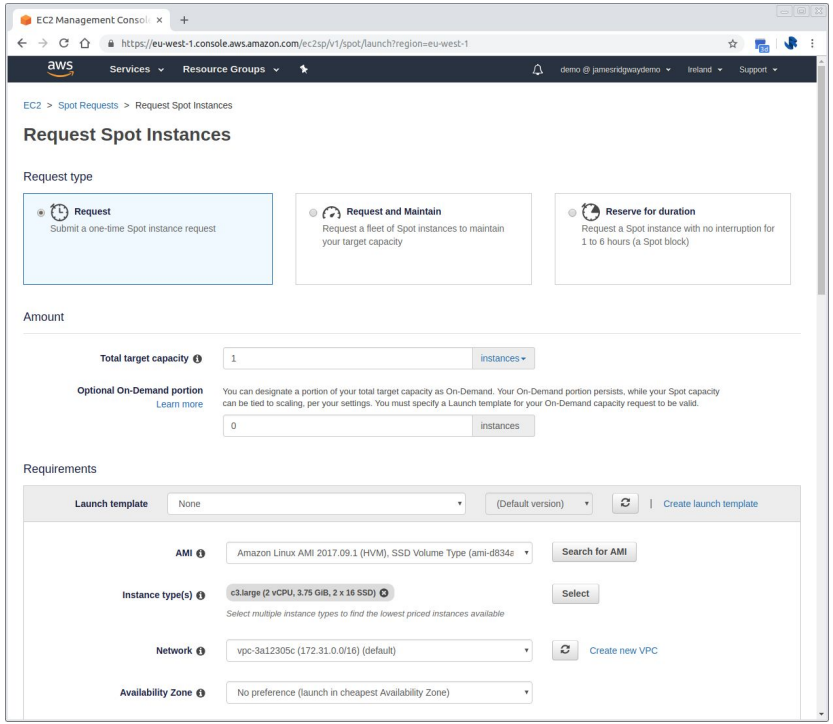
- Spare compute capacity at up to 90% off on-demand pricing
- Can be terminated with two minutes warning
- Different request types:
  - Request
  - Request and Maintain
  - Reserve for Duration
- Load balancer support
  - Classic
  - Application Load Balancer - Target Group



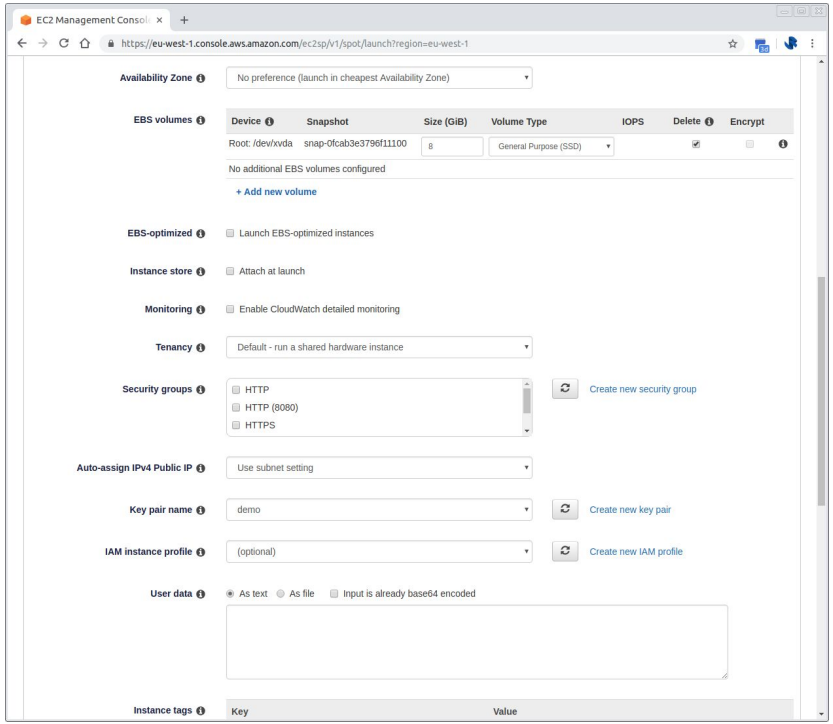
Spot Instance Price History



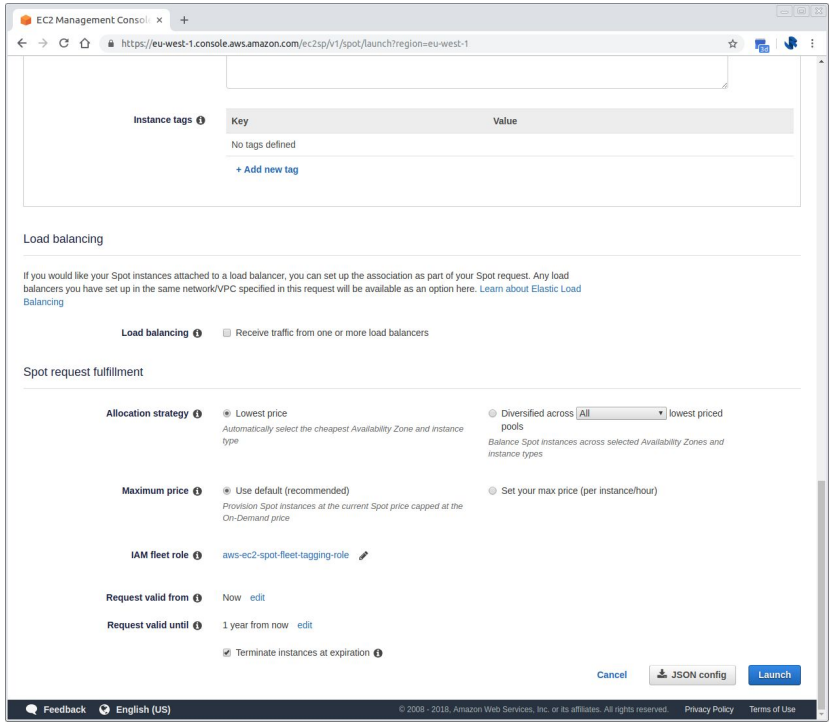
Requesting Spot Instances



Requesting Spot Instances



Requesting Spot Instances



Spot Blocks

- Launch spot instances that will run for a finite duration (1-6 hours)
- Pricing
  - Based on duration and capacity
  - Typically 30-45% less than on-demand price
  - Price per hour is fixed at launch, instance will not be terminated on price change
  - Partial hours are billed to the nearest second

Spot Fleet

- Instances are launched to meet target capacity
- One-time or persistent
- Can have on-demand as part of instance fleet
- Specify interrupt behaviour: hibernate, stop or terminate
    - Hibernate/stop only charges for preserved EBS volumes

Allocation Strategy

Strategies:

- `lowestPrice`
  - Launch to the instance pool with the lowest price
- `diversified`
  - Spot instances are distributed across all pools

Parameters:

- `InstancePoolsToUseCount`
  - Spot instances are distributed across the number of specified pools, only valid with lowestPrice

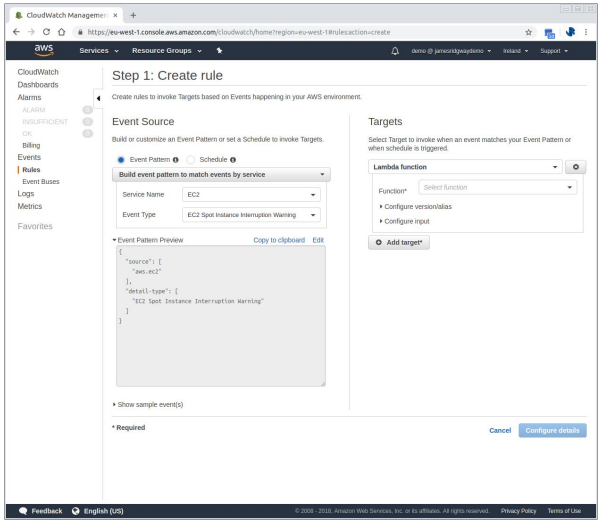
Spot Instance Interruption Notice

- A spot instance can be terminated with two minutes warning
- You control the interrupt behaviour (hibernate, stop, terminate)
- Reasons for interruption
  - Spot price exceeds maximum bid
  - Capacity shortage
  - Constraint - (e.g. no capacity within A-Z constraint)
    - All associated instances are terminated as a group

Spot Instance Interruption Notice

CloudWatch Event

Monitor for CloudWatch events and trigger target actions.




```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Spot Instance Interruption Warning",
  "source": "aws.ec2",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-east-2",
  "resources": ["arn:aws:ec2:us-east-2:123456789012:instance/i-1234567890abcdef0"],
  "detail": {
    "instance-id": "i-1234567890abcdef0",
    "instance-action": "action"
  }
}
```

Spot Instance Interruption Notice

Poll instance-action meta-data

AWS recommend polling every 5 seconds

Request:

```
[ec2-user ~]$ curl http://169.254.169.254/latest/meta-data/spot/instance-action
```

Response:

```
{"action": "terminate", "time": "2018-11-05T09:27:00Z"}
```

A Practical Example

Practical Example

- All server images should be built from scratch
- Use CI server to build whenever a change is pushed and automatically deploy
- Fully operate and run my web based projects on spot instances

Tooling


HashiCorp
Packer


SALTSTACK

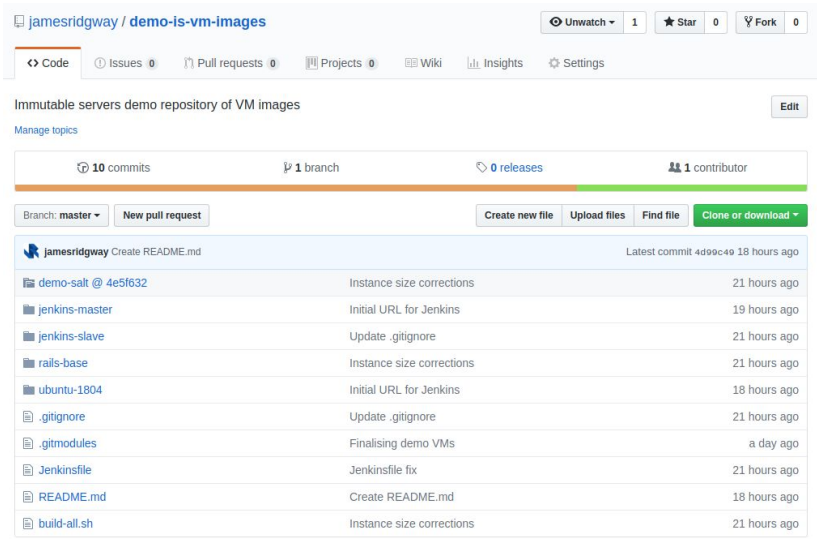

HashiCorp
Terraform


AWS Secrets
Manager

Building AMIs with Packer and SaltStack

Approach:

- Single repository with all base images
- Use my existing salt setup
  - Salt repository as a git submodule
- All images built with packer



Building AMIs with Packer and SaltStack

Lessons Learnt:

- Automatically pull latest parent image
  - `most_recent: true`
- Push EC2 user data into your base image
  - Use `/etc/rc.local` or `cloud-init`
- Layer images to reduce build time
- Tag your AMI with your git commit ID

Executable File | 6 lines (6 sloc) | 205 Bytes — Raw | Blame | History

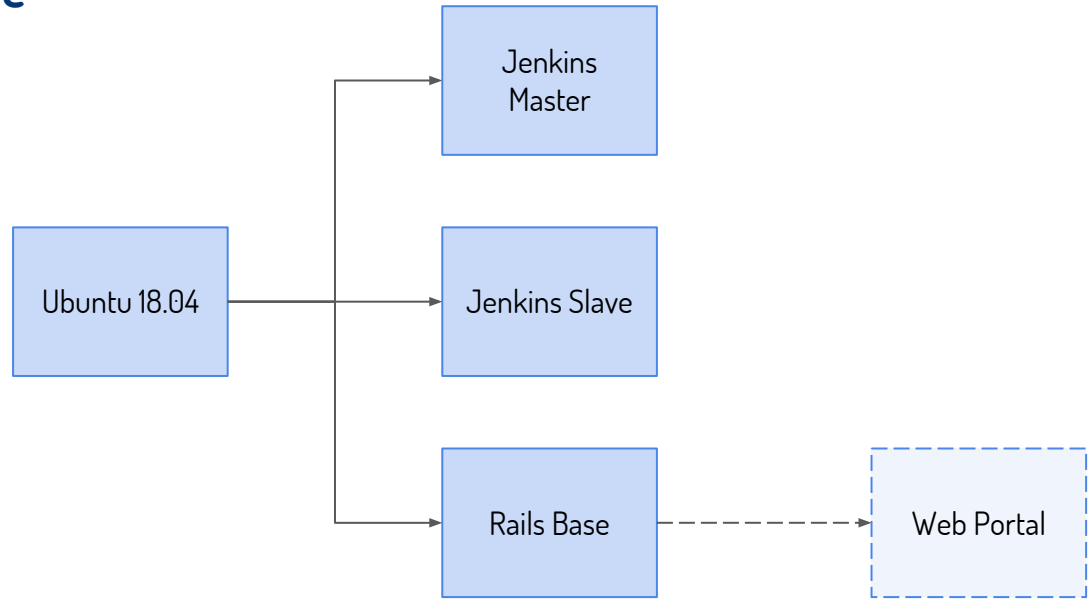
```
#!/bin/bash
set -e
cd "$(dirname "$0")" || exit 1
jq '.builders[0].tags.Commit = "'"$(git rev-parse HEAD)"'"' ubuntu.json > packer-versioned.json
packer build packer-versioned.json
rm packer-versioned.json
```

49 lines (49 sloc) | 1.1 KB — Raw | Blame | History

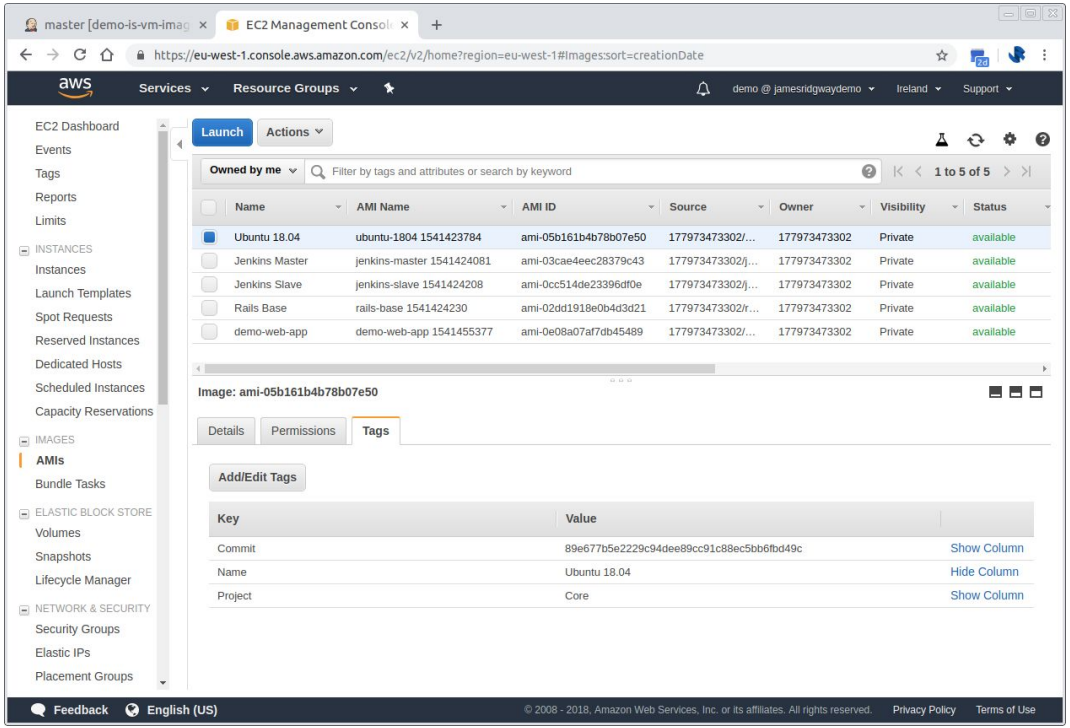
```
{
  "builders": [
    {
      "type": "amazon-ebs",
      "access_key": "{{user `aws_access_key`}}",
      "secret_key": "{{user `aws_secret_key`}}",
      "profile": "demo",
      "region": "eu-west-1",
      "instance_type": "t3.small",
      "ssh_username": "ubuntu",
      "source_ami_filter": {
        "filters": {
          "name": "ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-*"
        },
        "most_recent": true,
        "owners": ["099720109477"]
      },
      "ami_name": "ubuntu-1804 {{timestamp}}",
      "associate_public_ip_address": true,
      "tags": {
        "Name": "Ubuntu 18.04",
        "Project": "Core",
        "Commit": "unknown"
      }
    }
  ],
  "provisioners": [
    {
      "type": "file",
      "source": "../demo-salt/salt",
      "destination": "~/salt"
    },
    {
      "type": "shell",
      "execute_command": "echo 'vagrant' | {{.Vars}} sudo -S -E bash '{{.Path}}'",
      "scripts": [
        "scripts/init.sh"
      ]
    }
  ],
  "post-processors": [
    [
      {
        "output": "builds/{{.Provider}}-ubuntu1804.box",
        "type": "vagrant"
      }
    ]
  ]
}
```

Building AMIs with Packer and SaltStack

Structure



Building AMIs with Packer and SaltStack



Building a Jenkins Master

1. Install Java
2. Install Jenkins
3. Disable setup wizard
4. Configure Jenkins (using `init.groovy.d` scripts)
    - Any `*.groovy` script placed in `/var/lib/jenkins/init.groovy.d/` will be executed on boot

Building a Jenkins Master - `init.groovy.d` scripts

What I wanted to do:

1. Install plugins
2. Apply default security configuration
3. Create user accounts
4. Configure Jenkins to use spot instances
5. Enable GitHub webhook
6. Automatically create jobs for GitHub projects

Building a Jenkins Master - `init.groovy.d` scripts

Problem:

- All `*.groovy` scripts in `init.groovy.d` are loaded onto the classpath at the same point in time
- Trying to use code from plugins results in import errors:

```
import com.amazonaws.client.builder.AwsClientBuilder
import com.amazonaws.services.ec2.AmazonEC2
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder
...
```

Building a Jenkins Master - `init.groovy.d` scripts

Split up the steps:

- `init.groovy`
  a. Install plugins
  b. Apply default security configuration
  c. Rename *.txt files to *.groovy and trigger a Jenkins restart
- `aws.txt`
  a. Create user accounts
  b. Configure jenkins to use spot instances
- `github.txt`
  a. Enable GitHub webhook
- `githubrepos.txt`
  a. Automatically create jobs for GitHub projects

Solution:

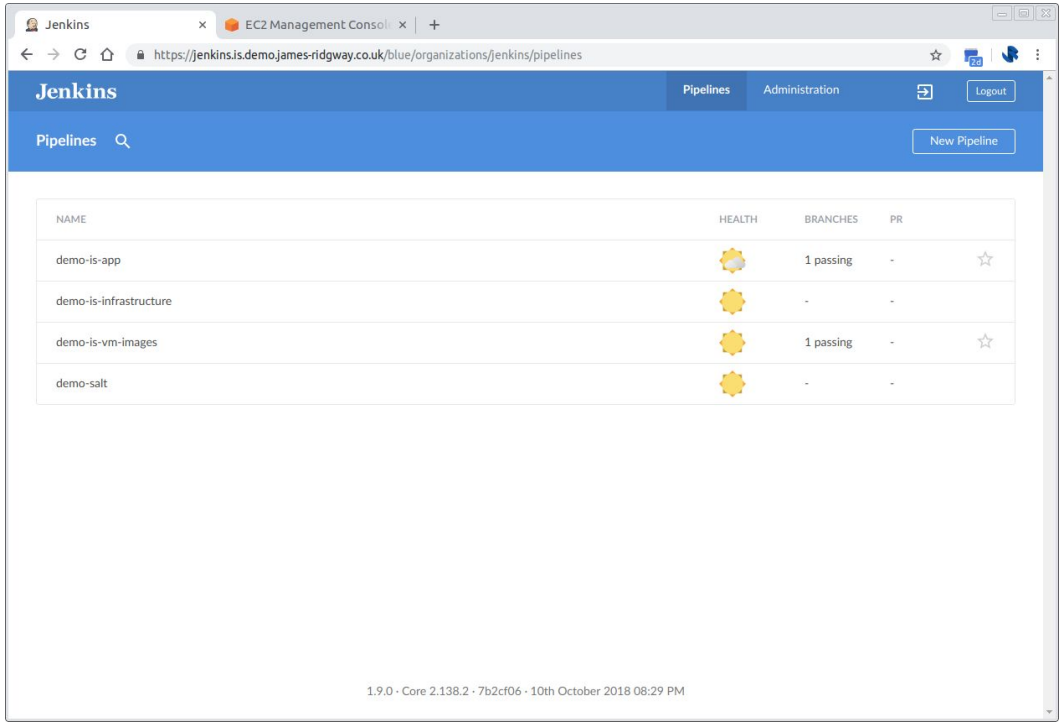
- Split steps into different scripts
- Initial script installs plugins and re-names other scripts so that these can be run on restart
  - Seems a little crude but this works flawlessly
- For ease write scripts to be idempotent

Building a Jenkins Master - `init.groovy.d` scripts

What did I learn:

- `init.groovy.d` scripts can be very powerful
- Not many documented examples
- When configuring the EC2 Spot plugin, don't use hard coded IDs
  - Find AMIs/SGs/IAM roles based on name, tags, etc
- A spot Jenkins master with spot slaves can be a great way to build projects with minimal cost
- Configuration should be external to the server image
  - Used AWS Secrets Manager for slave private key and key-value credentials

Jenkins on Spot Instances



Deploying Infrastructure with Terraform

- Data definitions pick up latest AMI images based on the name

```
data "aws_ami" "core_ami" {
  most_recent      = true
  filter {
    name   = "name"
    values = ["ubuntu-1804*"]
  }
  owners     = ["self"]
  tags {
    Project = "Core"
  }
}
```

- Spawned instances via `aws_spot_fleet_request`
- Deploying web based applications is really easy
  - Deploy your Application Load Balancer (ALB) and Target Groups
  - Let the spot fleet request deal with attaching and removing the spawned instances

Deploying Infrastructure with Terraform

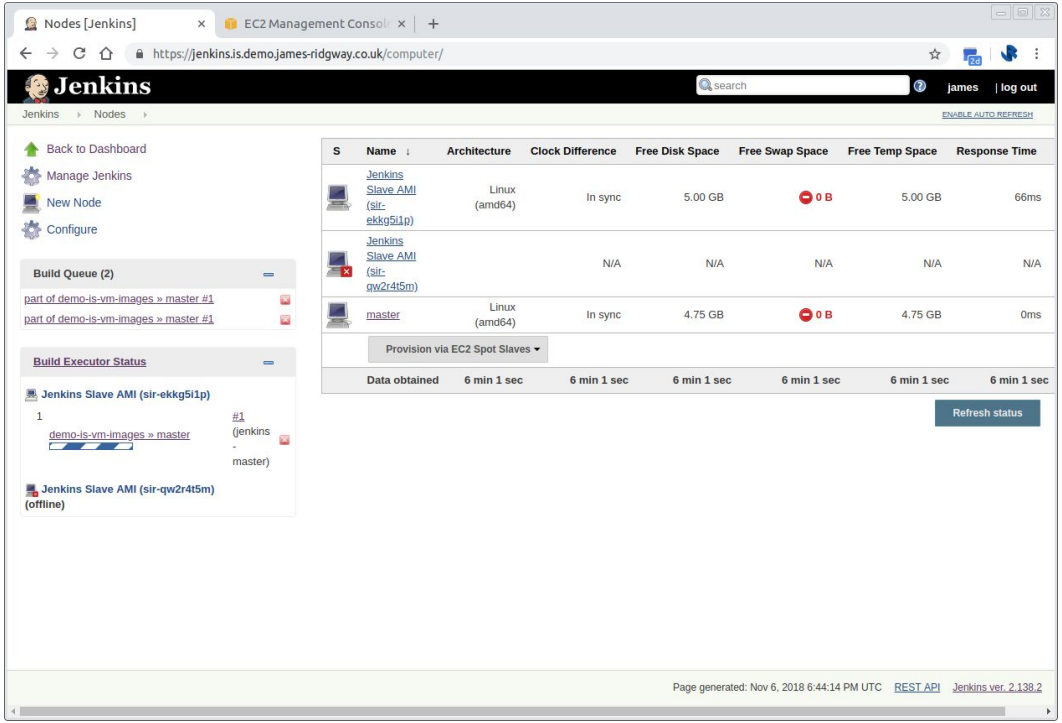
- Example of an `aws_spot_fleet_request`

```
resource "aws_spot_fleet_request" "jenkins-master" {
  iam_fleet_role      = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/aws-ec2-spot-fleet-tagging-role"
  allocation_strategy = "lowestPrice"
  target_capacity     = 1
  valid_until         = "2028-08-03T16:00:00Z"
  instance_interruption_behaviour = "stop"
  launch_specification {
    ami      = "${data.aws_ami.jenkins_master_ami.id}"
    instance_type = "t3.small"
    iam_instance_profile_arn = "${aws_iam_instance_profile.jenkins_ci_master.arn}"
    key_name = "demo"
    vpc_security_group_ids = ["${aws_security_group.ssh.id}",
                      "${aws_security_group.http_8080.id}",
                      "${data.aws_security_group.default.id}"]
    tags {
      Name = "Jenkins"
      Project = "Core"
    }
  }
  target_group_arns = ["${aws_lb_target_group.jenkins.arn}"]
}
```

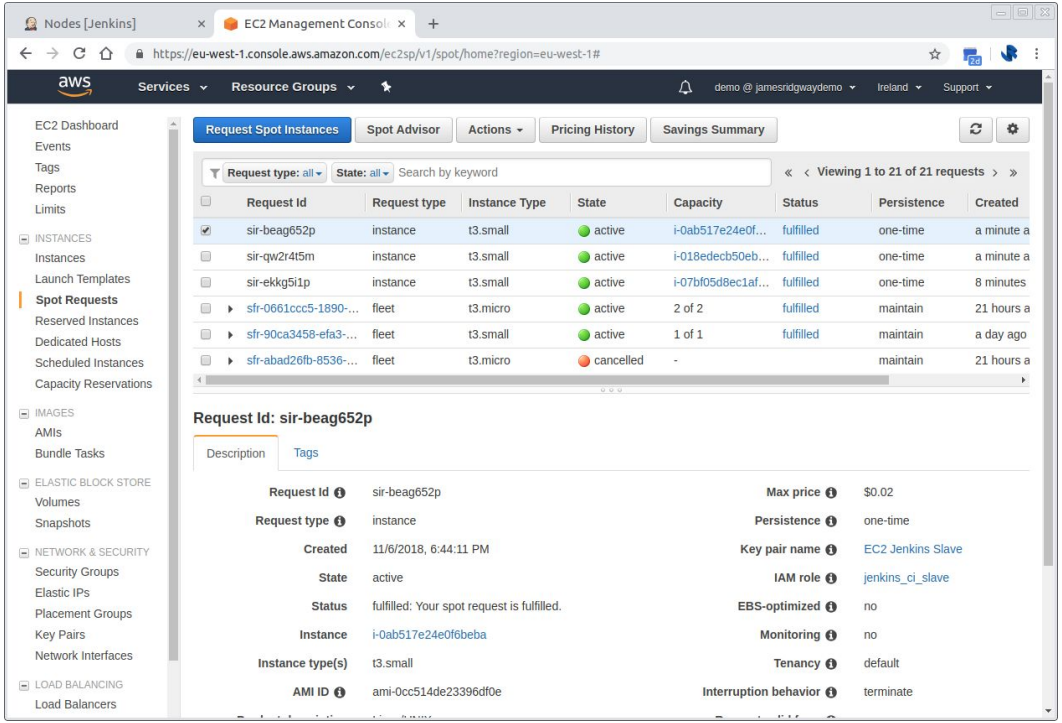
Deploying Infrastructure with Terraform

- Terraform deploys core infrastructure
  - (ALB, Jenkins Master, etc.)
- Application build pipelines produce AMIs and are responsible for spawning and managing spot requests - terraform doesn't need to know about these

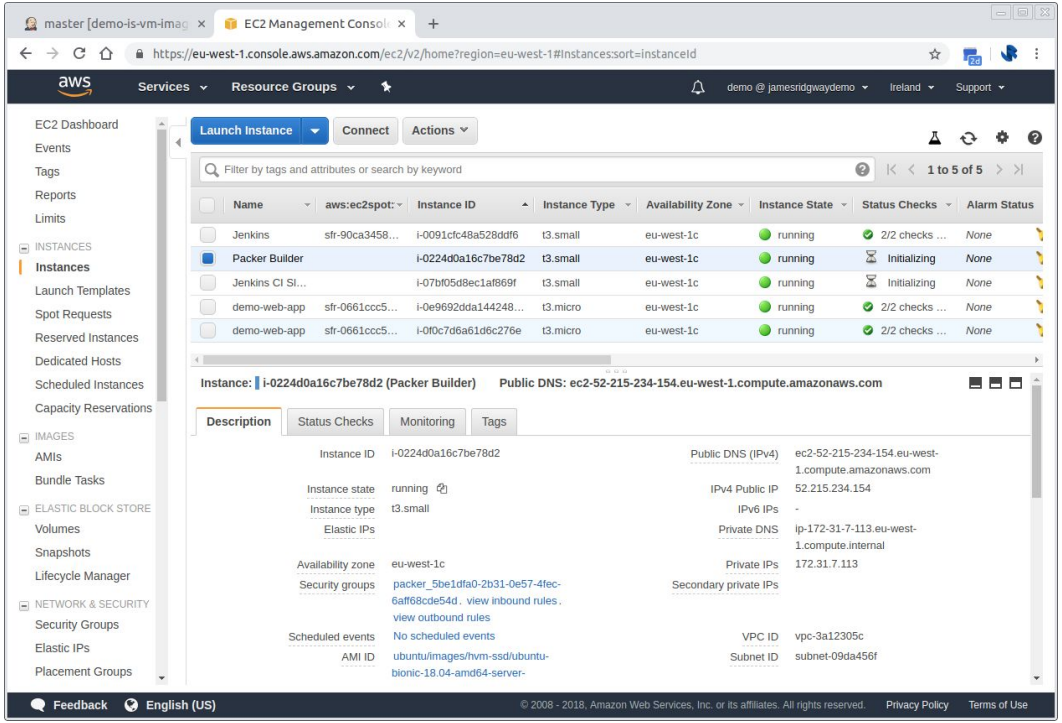
Jenkins on Spot Instances



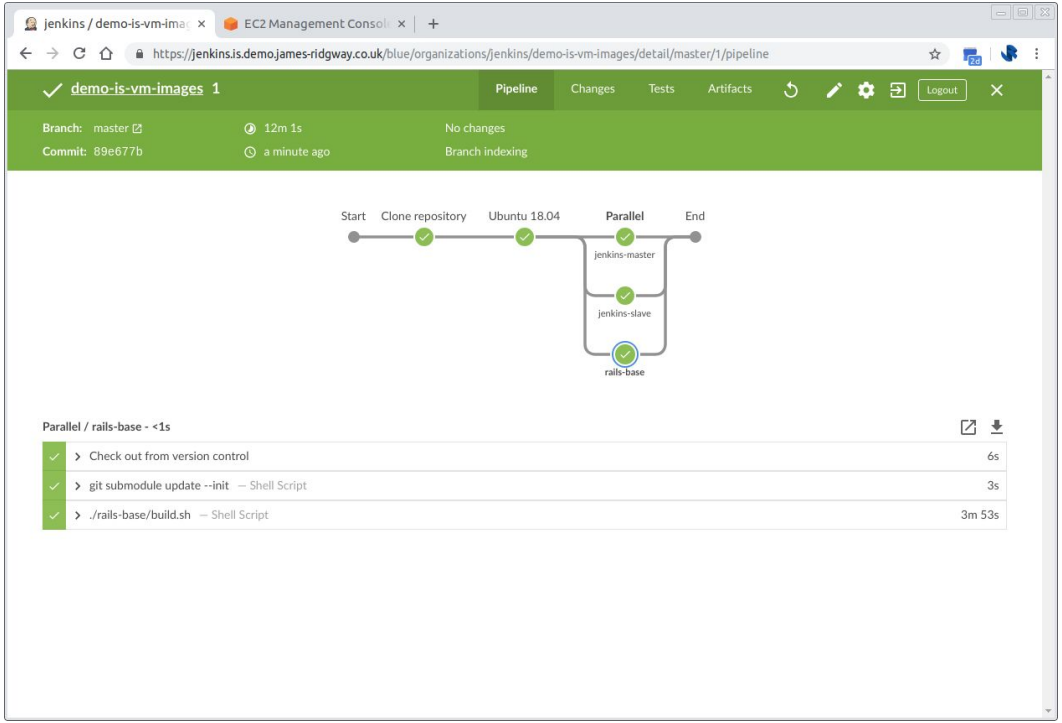
Jenkins on Spot Instances



Jenkins on Spot Instances



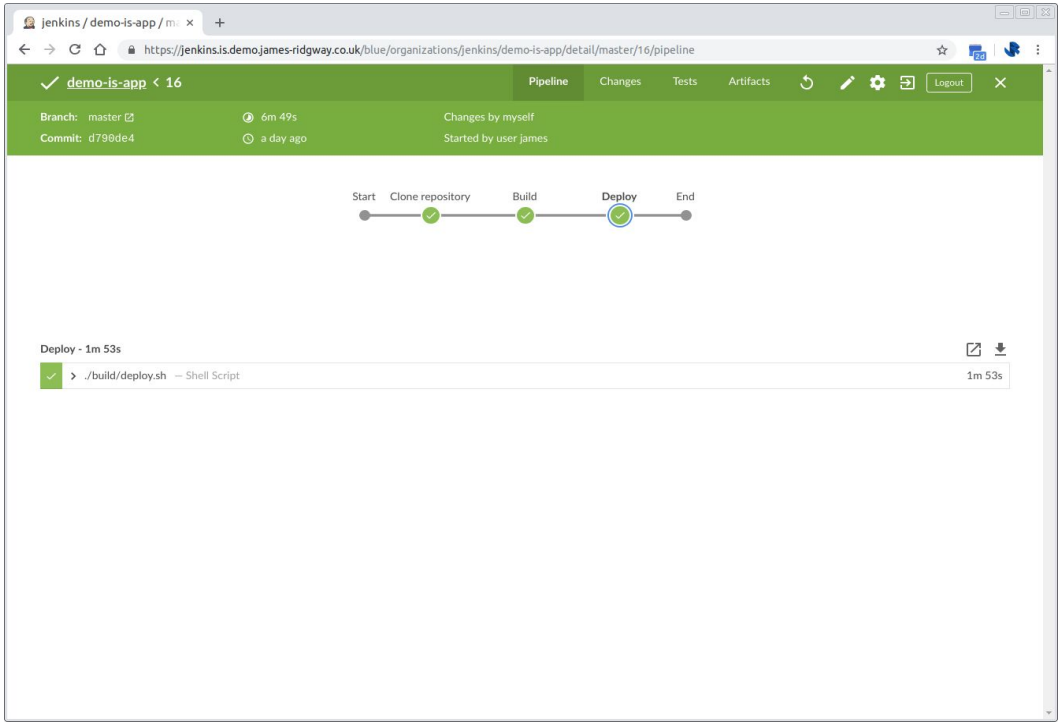
Jenkins on Spot Instances



Building the Application

- Simple rails demo application showing on-demand vs spot instance price
  - Permissions provided via IAM roles
- Built with Packer
  - Tests run within AMI as part of the packer build
  - Use manifest post processor to reliably identify identify AMI to deploy
- Separate build script

Building the Application



Building the Application – Packer

packer.json

```
{
  "builders": [
    {
      "type": "amazon-ebs",
      ...
    }
  ],
  "provisioners": [
    ...
  ],
  "post-processors": [
    [
      {
        "type": "manifest",
        "output": "manifest.json",
        "strip_path": true
      }
    ]
  ]
}
```

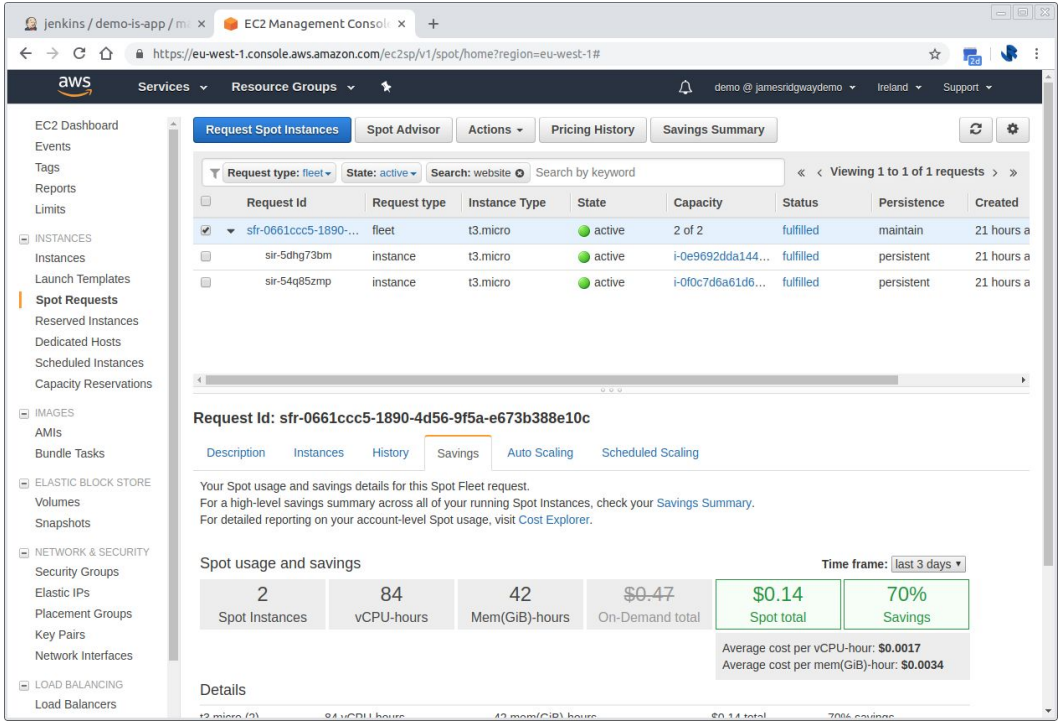
manifest.json

```
{
  "builds": [
    {
      "name": "amazon-ebs",
      "builder_type": "amazon-ebs",
      "build_time": 1541455668,
      "files": null,
      "artifact_id": "eu-west-1:ami-0e08a07af7db45489",
      "packer_run_uuid": "0e5155cc-d5aa-1eeb-db87-9884bd861"
    }
  ],
  "last_run_uuid": "0e5155cc-d5aa-1eeb-db87-9884bd861b08"
}
```

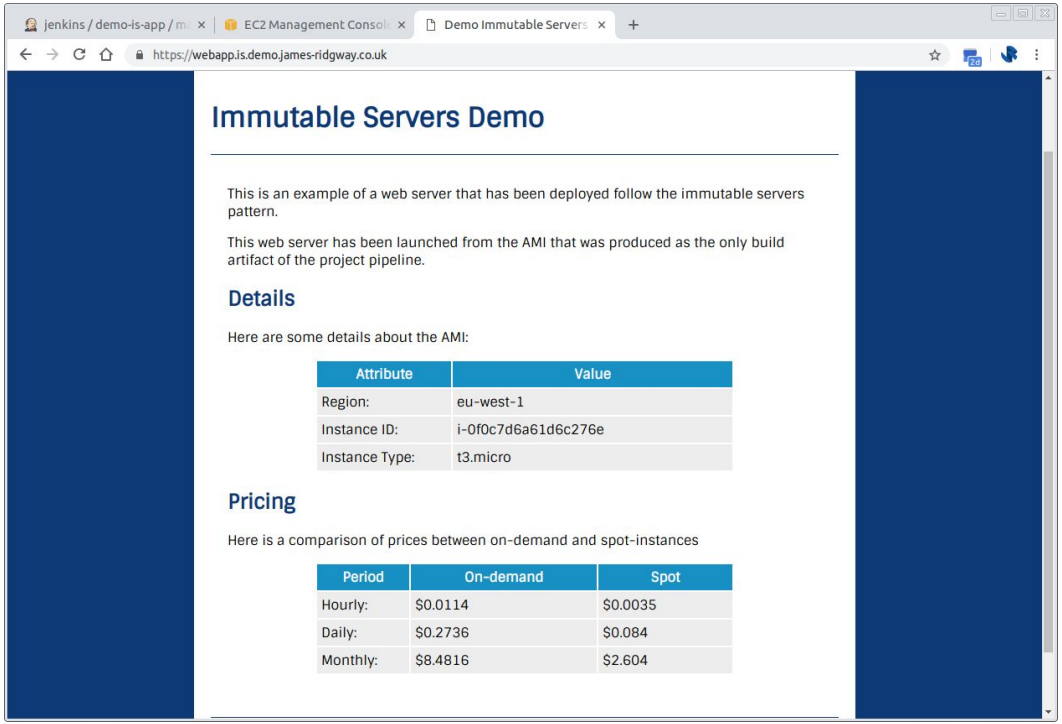
Building the Application – Deploy Process

1. Extract the AMI ID from the `manifest.json`
2. Search AWS by the *name* of assets:
    - ALB Target Group ARN, IAM Role ARNs, Security Group ARNs
3. Launch Spot Fleet Request
    - Wait for fulfilment
4. Inspect health of instances in LB target group
    - Cancel SFR on unhealthy instances, AWS will destroy the instances and remove them from the TG
5. Deployed!

An Application Running on Spot Instances



An Application Running on Spot Instances



Examples Available on GitHub

- Immutable servers demo repository of VM images
  - https://github.com/jamesridgway/demo-is-vm-images
- Demo salt
  - https://github.com/jamesridgway/demo-salt
- Immutable servers infrastructure demo
  - https://github.com/jamesridgway/demo-is-infrastructure
- Immutable servers web app demo
  - https://github.com/jamesridgway/demo-is-app

What Did I Learn?

- Immutable Servers have similar advantages and disadvantages to containers
- Immutable Servers can encourage good design of systems/deployment processes
- Spot instances are simple to use and are cheap!
    - Aggressively tag to monitor your costs
    - Embed version information in AMI
- Jenkins can be fully automated
- Minimise start time, bake more into the base image

Thank you

Any questions?

@james_ridgway

jamesridgway

james-ridgway.co.uk

Slides

jmsr.io/PSWnJz

Immutable Servers

Building a deployment pipeline and deploying to EC2 Spot Instances