

Immutable Servers

Building a deployment pipeline and deploying to EC2 Spot Instances

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

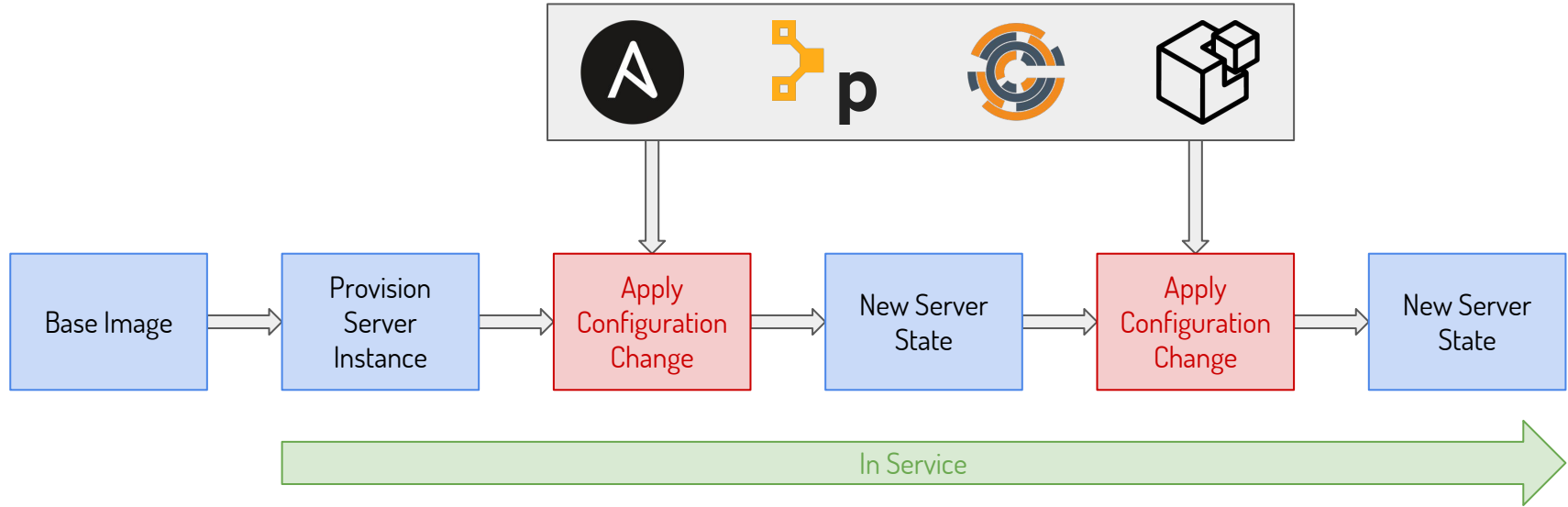
Definition:

- 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

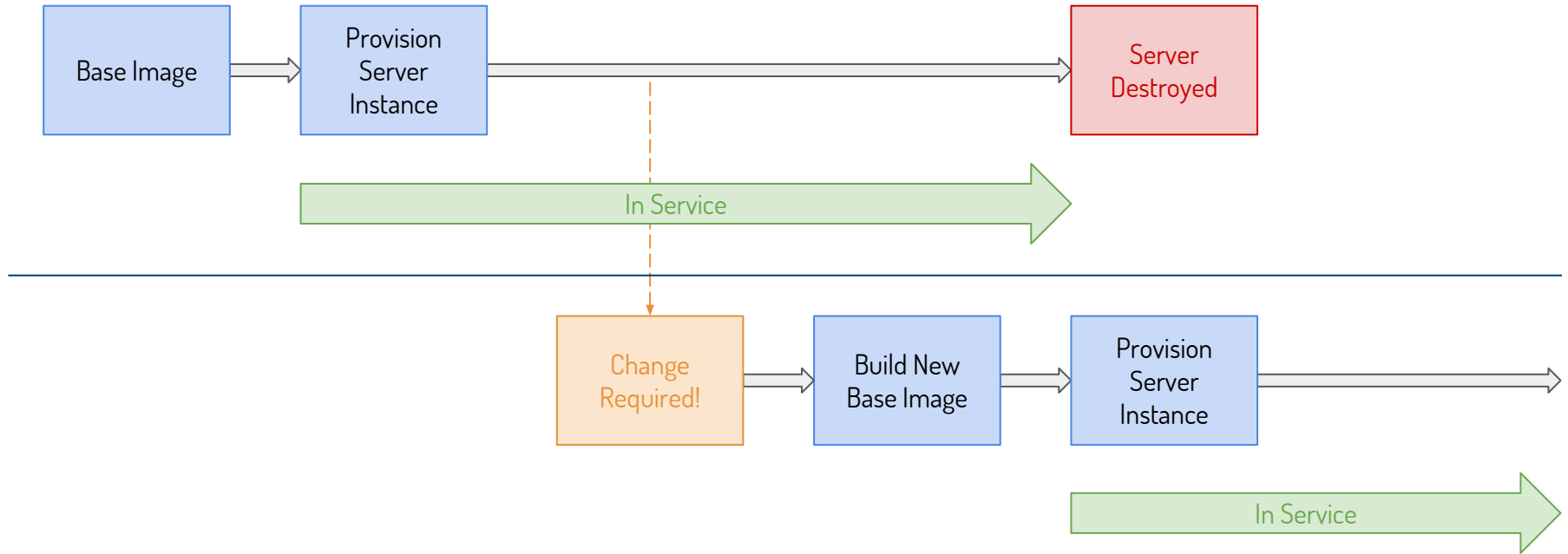
Advantages:

- 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 surface 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

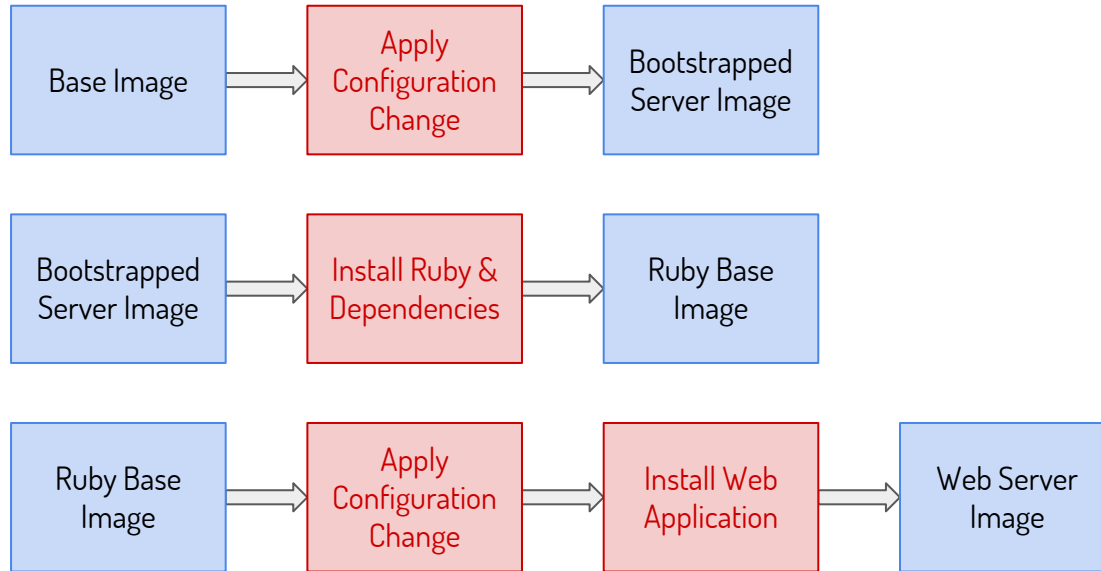
“In the old way of doing things, we treat our servers like pets, for example Bob the mail server. If Bob goes down, it’s all hands on deck. The CEO can’t get his email and it’s the end of the world. In the new way, servers are numbered, like cattle in a herd. For example, www001 to www100. When one server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

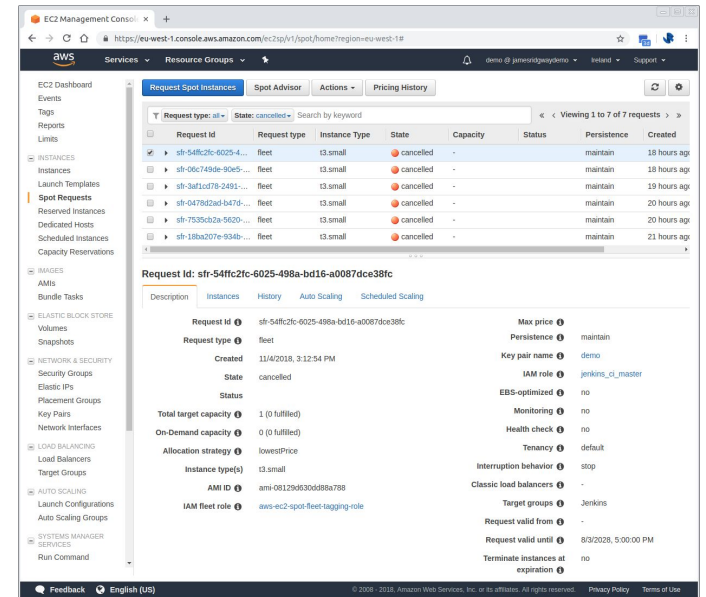
- Pay per hour at the full rate

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

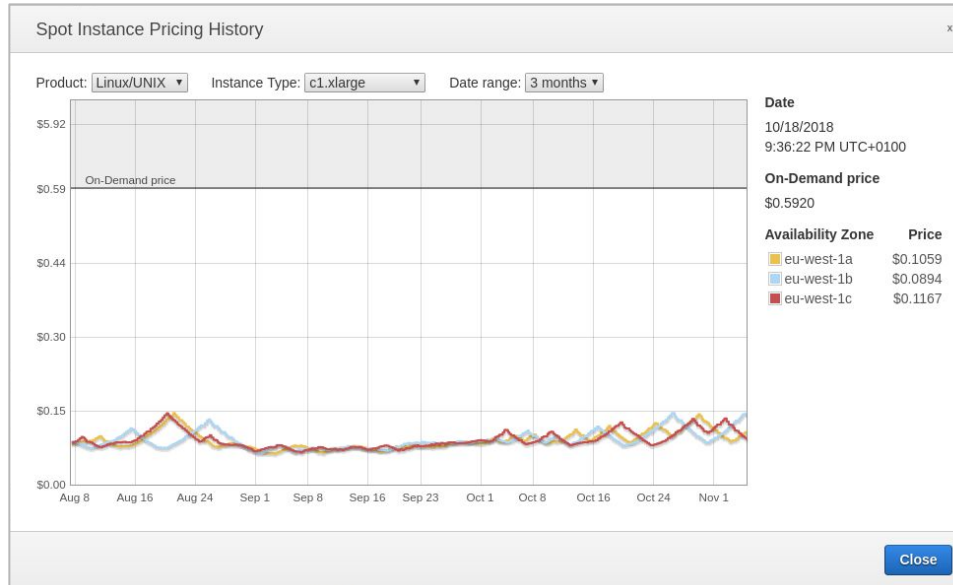


The screenshot displays the AWS Management Console interface for Spot Instance requests. The main view shows a table of requests with columns for Request ID, Request type, Instance Type, State, Capacity, Status, Persistence, and Created. All listed requests are in a 'cancelled' state.

Request ID	Request type	Instance Type	State	Capacity	Status	Persistence	Created
sfr-54ffc2fc-6025-4...	fleet	t3.small	cancelled	-	-	maintain	18 hours ago
sfr-06c749de-9065-...	fleet	t3.small	cancelled	-	-	maintain	18 hours ago
sfr-3af1a478-2495-...	fleet	t3.small	cancelled	-	-	maintain	19 hours ago
sfr-0478d2ad-b47d-...	fleet	t3.small	cancelled	-	-	maintain	20 hours ago
sfr-7535cb2a-5620-...	fleet	t3.small	cancelled	-	-	maintain	20 hours ago
sfr-18fa207e-934b-...	fleet	t3.small	cancelled	-	-	maintain	21 hours ago

Below the table, the details for the selected request (Request ID: sfr-54ffc2fc-6025-498a-bd16-a0087dce38fc) are displayed. The request is in a 'cancelled' state, created on 11/4/2018 at 3:12:54 PM. Key configuration details include: Instance Type (t3.small), AMI ID (ami-08129d030cd98a788), IAM fleet role (aws-ec2-spot-fleet-tagging-role), and a persistence policy of 'maintain'.

Spot Instance Price History



Requesting Spot Instances

The screenshot shows the AWS Management Console interface for requesting Spot Instances. The page is titled "Request Spot Instances" and is located under the "EC2 > Spot Requests > Request Spot Instances" breadcrumb. The interface is divided into several sections:

- Request type:** Three options are available: "Request" (Submit a one-time Spot instance request), "Request and Maintain" (Request a fleet of Spot instances to maintain your target capacity), and "Reserve for duration" (Request a Spot instance with no interruption for 1 to 6 hours (a Spot block)). The "Request" option is selected.
- Amount:** Two input fields are present: "Total target capacity" (set to 1) and "Optional On-Demand portion" (set to 0). A note explains that the On-Demand portion can be designated as On-Demand and is tied to scaling.
- Requirements:** This section includes:
 - Launch template:** Set to "None" with a "Create launch template" button.
 - AMI:** Set to "Amazon Linux AMI 2017.09.1 (HVM), SSD Volume Type (ami-d834e)" with a "Search for AMI" button.
 - Instance type(s):** Set to "c3.large (2 vCPU, 3.75 GiB, 2 x 16 SSD)" with a "Select" button. A note below says "Select multiple instance types to find the lowest priced instances available".
 - Network:** Set to "vpc-3a12305c (172.31.0.0/16) (default)" with a "Create new VPC" button.
 - Availability Zone:** Set to "No preference (launch in cheapest Availability Zone)".

Requesting Spot Instances

EC2 Management Console x +

https://eu-west-1.console.aws.amazon.com/ec2sp/v1/spot/launch?region=eu-west-1

Availability Zone ⓘ No preference (launch in cheapest Availability Zone)

EBS volumes ⓘ

Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete	Encrypt
Root: /dev/xvda	snap-0cab3e3796f11100	8	General Purpose (SSD)		<input checked="" type="checkbox"/>	<input type="checkbox"/>

No additional EBS volumes configured

[+ Add new volume](#)

EBS-optimized ⓘ Launch EBS-optimized instances

Instance store ⓘ Attach at launch

Monitoring ⓘ Enable CloudWatch detailed monitoring

Tenancy ⓘ Default - run a shared hardware instance

Security groups ⓘ

- HTTP
- HTTP (8080)
- HTTPS

[Create new security group](#)

Auto-assign IPv4 Public IP ⓘ Use subnet setting

Key pair name ⓘ demo [Create new key pair](#)

IAM instance profile ⓘ (optional) [Create new IAM profile](#)

User data ⓘ As text As file Input is already base64 encoded

Instance tags ⓘ

Key	Value
-----	-------

Requesting Spot Instances

The screenshot displays the AWS Management Console interface for launching Spot Instances. The browser address bar shows the URL: `https://eu-west-1.console.aws.amazon.com/ec2sp/v1/spot/launch?region=eu-west-1`.

Instance tags

Key	Value
No tags defined	

[+ Add new tag](#)

Load balancing

If you would like your Spot instances attached to a load balancer, you can set up the association as part of your Spot request. Any load balancers you have set up in the same network/VPC specified in this request will be available as an option here. [Learn about Elastic Load Balancing](#)

Load balancing Receive traffic from one or more load balancers

Spot request fulfillment

Allocation strategy

- Lowest price**
Automatically select the cheapest Availability Zone and instance type
- Diversified across [All] lowest priced pools**
Balance Spot instances across selected Availability Zones and instance types

Maximum price

- Use default (recommended)**
Provision Spot instances at the current Spot price capped at the On-Demand price
- Set your max price (per instancehour)**

IAM fleet role `aws-ec2-spot-fleet-tagging-role` [edit](#)

Request valid from `Now` [edit](#)

Request valid until `1 year from now` [edit](#)

Terminate instances at expiration

[Cancel](#) [JSON config](#) [Launch](#)

Feedback English (US) © 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

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

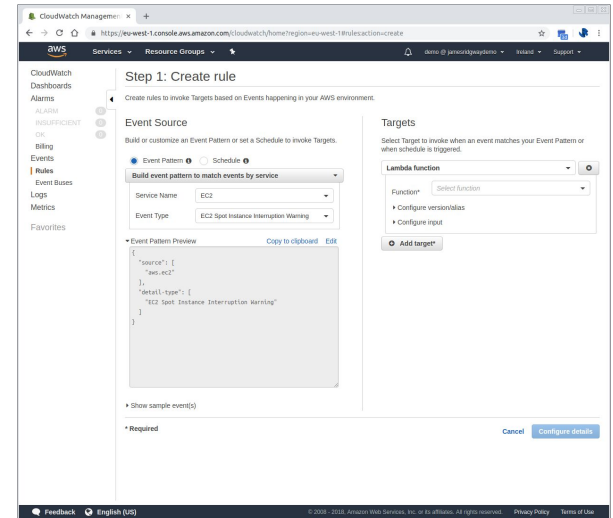
- A spot instance can be terminated with two minutes warning
- You control the interrupt behaviour (hibernate, stop, terminate)
- Reasons for interruption
 - Spot price exceeds maximum bid
 - Capacity shortage
 - Constraint - (e.g. no capacity within A-Z constraint)
 - All associated instances are terminated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



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

The screenshot shows a GitHub repository page for 'jamesridgway / demo-is-vm-images'. The repository is described as 'Immutable servers demo repository of VM images'. It has 10 commits, 1 branch, 0 releases, and 1 contributor. The repository structure is as follows:

File/Directory	Description	Last Commit
demo-salt @ 4e5f632	Instance size corrections	21 hours ago
jenkins-master	Initial URL for Jenkins	19 hours ago
jenkins-slave	Update .gitignore	21 hours ago
rails-base	Instance size corrections	21 hours ago
ubuntu-1804	Initial URL for Jenkins	18 hours ago
.gitignore	Update .gitignore	21 hours ago
.gitmodules	Finalising demo VMs	a day ago
Jenkinsfile	Jenkinsfile fix	21 hours ago
README.md	Create README.md	18 hours ago
build-all.sh	Instance size corrections	21 hours ago

Building AMIs with Packer and SaltStack

Lessons Learnt:

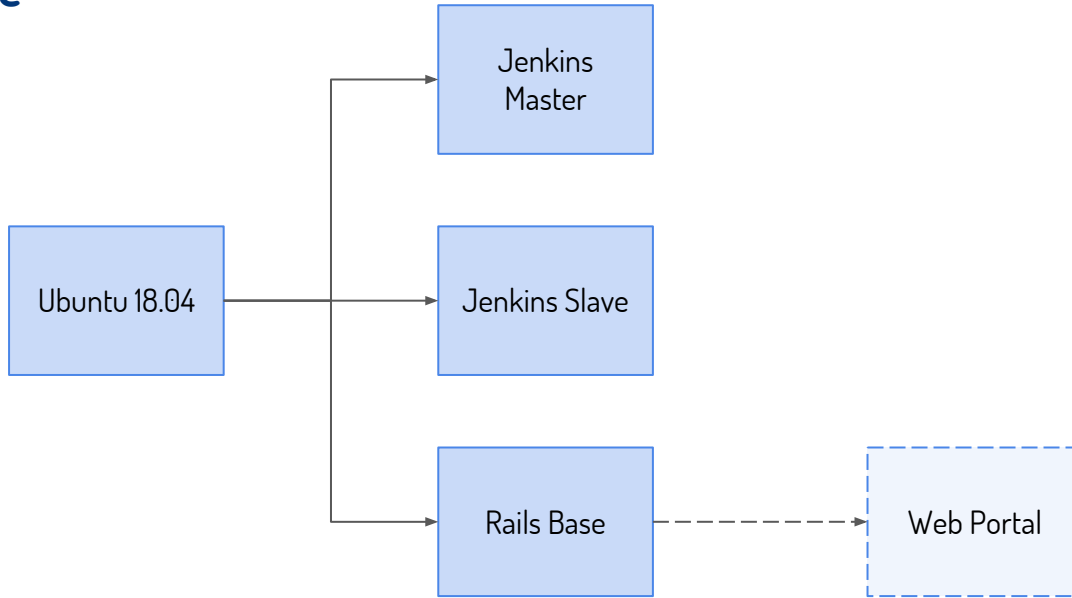
- Automatically pull latest parent image
 - `most_recent: true`
- Push EC2 user data into your base image
 - Use `/etc/rc.local` or `cloud-init`
- Layer images to reduce build time
- Tag your AMI with your git commit ID

```
Executable File | 6 lines (6 sloc) | 205 Bytes
1 #!/bin/bash
2 set -e
3 cd "$(dirname "$0")" || exit 1
4 jq '.builders[0].tags.Commit = "'$(git rev-parse HEAD)'"' ubuntu.json > packer-versioned.json
5 packer build packer-versioned.json
6 rm packer-versioned.json
```

```
49 lines (49 sloc) | 1.1 KB
1 {
2   "builders": [
3     {
4       "type": "amazon-ec2",
5       "access_key": "{{user `aws_access_key`}}",
6       "secret_key": "{{user `aws_secret_key`}}",
7       "profile": "demo",
8       "region": "eu-west-1",
9       "instance_type": "t3.small",
10      "ssh_username": "ubuntu",
11      "source_ami_filter": {
12        "filters": {
13          "name": "ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-*"
14        },
15        "most_recent": true,
16        "owners": ["099720109477"]
17      },
18      "ami_name": "ubuntu-1804-{{timestamp}}",
19      "associate_public_ip_address": true,
20      "tags": {
21        "Name": "Ubuntu 18.04",
22        "Project": "Core",
23        "Commit": "unknown"
24      }
25    },
26  ],
27  "provisioners": [
28    {
29      "type": "file",
30      "source": "../demo-salt/salt",
31      "destination": "~/salt"
32    },
33    {
34      "type": "shell",
35      "execute_command": "echo 'vagrant' | {{.Vars}} sudo -S -E bash '{{.Path}}'",
36      "scripts": [
37        "scripts/init.sh"
38      ]
39    },
40  ],
41  "post-processors": [
42    {
43      "output": "builds/{{.Provider}}-ubuntu1804.box",
44      "type": "vagrant"
45    }
46  ]
47 }
48 ]
49 }
```

Building AMIs with Packer and SaltStack

Structure



Building AMIs with Packer and SaltStack

The screenshot displays the AWS Management Console interface. The left-hand navigation pane shows the 'AMIs' section under 'IMAGES'. The main content area shows a list of AMIs with the following data:

Name	AMI Name	AMI ID	Source	Owner	Visibility	Status
Ubuntu 18.04	ubuntu-1804 1541423784	ami-05b161b4b78b07e50	177973473302/...	177973473302	Private	available
Jenkins Master	jenkins-master 1541424081	ami-03cae4eec28379c43	177973473302/j...	177973473302	Private	available
Jenkins Slave	jenkins-slave 1541424208	ami-0cc514de23396d0e	177973473302/j...	177973473302	Private	available
Rails Base	rails-base 1541424230	ami-02dd1918e0b4d3d21	177973473302/r...	177973473302	Private	available
demo-web-app	demo-web-app 1541455377	ami-0e08a07af7db45489	177973473302/...	177973473302	Private	available

The selected AMI, 'ami-05b161b4b78b07e50', is shown in the 'Tags' tab with the following details:

Key	Value
Commit	89e677b5e2229c94dee89cc91c88ec5bb6bd49c
Name	Ubuntu 18.04
Project	Core

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:

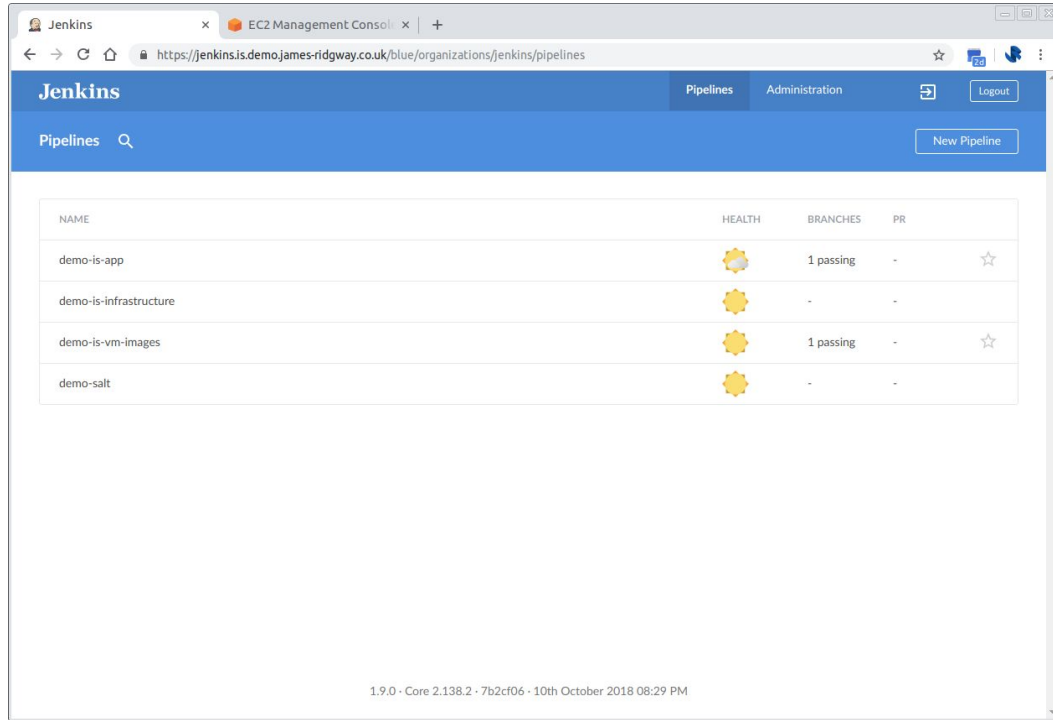
- Split steps into different scripts
- Initial script installs plugins and re-names other scripts so that these can be run on restart
 - Seems a little crude but this works flawlessly
- For ease write scripts to be idempotent

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



The screenshot shows the Jenkins web interface in a browser window. The address bar displays the URL: `https://jenkins.is.demojames-ridgway.co.uk/blue/organizations/jenkins/pipelines`. The page header includes the Jenkins logo, navigation tabs for "Pipelines" and "Administration", and a "Logout" button. Below the header, there is a search bar for pipelines and a "New Pipeline" button. The main content area features a table with the following data:

NAME	HEALTH	BRANCHES	PR
demo-is-app		1 passing	-
demo-is-infrastructure		-	-
demo-is-vm-images		1 passing	-
demo-salt		-	-

At the bottom of the page, the version information is displayed: "1.9.0 - Core 2.138.2 - 7b2cf06 - 10th October 2018 08:29 PM".

Deploying Infrastructure with Terraform

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

```
1 data "aws_ami" "core_ami" {
2   most_recent = true
3   filter {
4     name = "name"
5     values = ["ubuntu-1804*"]
6   }
7   owners = ["self"]
8   tags {
9     Project = "Core"
10  }
11 }
```

- 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`

```
13 resource "aws_spot_fleet_request" "jenkins-master" {
14     iam_fleet_role      = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/aws-ec2-spot-fleet-tagging-role"
15     allocation_strategy = "lowestPrice"
16     target_capacity     = 1
17     valid_until        = "2028-08-03T16:00:00Z"
18     instance_interruption_behaviour = "stop"
19     launch_specification {
20         ami           = "${data.aws_ami.jenkins_master.ami.id}"
21         instance_type = "t3.small"
22         iam_instance_profile_arn = "${aws_iam_instance_profile.jenkins_ci_master.arn}"
23         key_name      = "demo"
24         vpc_security_group_ids = ["${aws_security_group.ssh.id}",
25                                 "${aws_security_group.http_8080.id}",
26                                 "${data.aws_security_group.default.id}"]
27         tags {
28             Name = "Jenkins"
29             Project = "Core"
30         }
31     }
32     target_group_arns = ["${aws_lb_target_group.jenkins.arn}"]
33 }
```

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

Nodes [Jenkins] x EC2 Management Console x +

https://jenkins.is.demo.james-ridgway.co.uk/computer/

Jenkins search james | log out

ENABLE AUTO REFRESH

Back to Dashboard
Manage Jenkins
New Node
Configure

Build Queue (2)

- part of demo-is-vm-images » master #1
- part of demo-is-vm-images » master #1

Build Executor Status

- Jenkins Slave AMI (sir-ekkg51p)
 - 1 #1 (jenkins - master)
- Jenkins Slave AMI (sir-qw2r4t5m) (offline)

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Jenkins Slave AMI (sir-ekkg51p)	Linux (amd64)	In sync	5.00 GB	0 B	5.00 GB	66ms
	Jenkins Slave AMI (sir-qw2r4t5m)		N/A	N/A	N/A	N/A	N/A
	master	Linux (amd64)	In sync	4.75 GB	0 B	4.75 GB	0ms

Provision via EC2 Spot Slaves

Data obtained	6 min 1 sec	6 min 1 sec	6 min 1 sec	6 min 1 sec	6 min 1 sec	6 min 1 sec
---------------	-------------	-------------	-------------	-------------	-------------	-------------

Refresh status

Page generated: Nov 6, 2018 6:44:14 PM UTC [REST API](#) [Jenkins ver. 2.138.2](#)

Jenkins on Spot Instances

The screenshot shows the AWS Management Console interface for Spot Instance requests. The left sidebar contains navigation options like EC2 Dashboard, INSTANCES, and ELASTIC BLOCK STORE. The main content area is titled 'Request Spot Instances' and includes a search bar and a table of requests.

Request Id	Request type	Instance Type	State	Capacity	Status	Persistence	Created
<input checked="" type="checkbox"/> sir-beag652p	instance	t3.small	active	i-0ab517e24e0f...	fulfilled	one-time	a minute a
<input type="checkbox"/> sir-qw2r4t5m	instance	t3.small	active	i-018edecb50eb...	fulfilled	one-time	a minute a
<input type="checkbox"/> sir-ekkg51p	instance	t3.small	active	i-07bf05d8ec1af...	fulfilled	one-time	8 minutes
<input type="checkbox"/> ▶ sfr-0661ccc5-1890-...	fleet	t3.micro	active	2 of 2	fulfilled	maintain	21 hours a
<input type="checkbox"/> ▶ sfr-90ca3458-efa3-...	fleet	t3.small	active	1 of 1	fulfilled	maintain	a day ago
<input type="checkbox"/> ▶ sfr-abad26fb-8536-...	fleet	t3.micro	cancelled	-	-	maintain	21 hours a

Request Id: sir-beag652p

Description | Tags

Request Id	sir-beag652p	Max price	\$0.02
Request type	instance	Persistence	one-time
Created	11/6/2018, 6:44:11 PM	Key pair name	EC2 Jenkins Slave
State	active	IAM role	jenkins_ci_slave
Status	fulfilled: Your spot request is fulfilled.	EBS-optimized	no
Instance	i-0ab517e24e0f6beba	Monitoring	no
Instance type(s)	t3.small	Tenancy	default
AMI ID	ami-0cc514de23396df0e	Interruption behavior	terminate

Jenkins on Spot Instances

The screenshot displays the AWS Management Console interface. The left-hand navigation pane includes sections for INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The main content area shows a table of EC2 instances with columns for Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, and Alarm Status. The 'Packer Builder' instance is selected and highlighted in blue.

Name	aws:ec2spot:	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
Jenkins	sfr-90ca3458...	i-0091cfc48a528ddf6	t3.small	eu-west-1c	running	2/2 checks ...	None
Packer Builder		i-0224d0a16c7be78d2	t3.small	eu-west-1c	running	Initializing	None
Jenkins CI S...		i-07b05d8ec1af869f	t3.small	eu-west-1c	running	Initializing	None
demo-web-app	sfr-0661ccc5...	i-0e9692dda144248...	t3.micro	eu-west-1c	running	2/2 checks ...	None
demo-web-app	sfr-0661ccc5...	i-0f0c7d6a61d6c276e	t3.micro	eu-west-1c	running	2/2 checks ...	None

Instance: **i-0224d0a16c7be78d2 (Packer Builder)** Public DNS: ec2-52-215-234-154.eu-west-1.compute.amazonaws.com

Description | Status Checks | Monitoring | Tags

Instance ID	i-0224d0a16c7be78d2	Public DNS (IPv4)	ec2-52-215-234-154.eu-west-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	52.215.234.154
Instance type	t3.small	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-7-113.eu-west-1.compute.internal
Availability zone	eu-west-1c	Private IPs	172.31.7.113
Security groups	packer_5be1dfa0-2b31-0e57-4fec-6aff68cde54d. view inbound rules. view outbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-3a12305c
AMI ID	ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-	Subnet ID	subnet-09da456f

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Jenkins on Spot Instances

jenkins / demo-is-vm-ima x EC2 Management Consol x +

https://jenkins.is.demojames-ridgway.co.uk/blue/organizations/jenkins/demo-is-vm-images/detail/master/1/pipeline

✓ demo-is-vm-images 1 Pipeline Changes Tests Artifacts Logout X

Branch: master 12m 1s No changes
Commit: 89e677b a minute ago Branch indexing

Start Clone repository Ubuntu 18.04 Parallel End

jenkins-master
jenkins-slave
rails-base

Parallel / rails-base - <1s

- ✓ > Check out from version control 6s
- ✓ > git submodule update --init -- Shell Script 3s
- ✓ > ./rails-base/build.sh -- Shell Script 3m 53s

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

The screenshot shows the Jenkins web interface for a pipeline named 'demo-is-app'. The pipeline is currently running on the 'master' branch, commit 'd798de4', which was pushed '6m 49s' ago. The pipeline consists of five stages: Start, Clone repository, Build, Deploy, and End. All stages have completed successfully, indicated by green checkmarks. The 'Deploy' stage is currently selected and highlighted with a blue circle. Below the pipeline graph, the 'Deploy' stage's execution details are shown, including a shell script command: `./build/deploy.sh`.

jenkins / demo-is-app / master / 16

https://jenkins.is.demojames-ridgway.co.uk/blue/organizations/jenkins/demo-is-app/detail/master/16/pipeline

demo-is-app < 16

Pipeline Changes Tests Artifacts Logout

Branch: master 6m 49s Changes by myself

Commit: d798de4 1 day ago Started by user james

Start Clone repository Build Deploy End

Deploy - 1m 53s

```
✓ > ./build/deploy.sh - Shell Script 1m 53s
```

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

The screenshot displays the AWS Management Console interface for Spot Instance requests. The left-hand navigation pane shows various AWS services, with 'Spot Requests' highlighted under the 'INSTANCES' category. The main content area shows a list of Spot Instance requests, with one selected. Below the list, the 'Request Id: sfr-0661ccc5-1890-4d56-9f5a-e673b388e10c' is shown, along with tabs for 'Description', 'Instances', 'History', 'Savings', 'Auto Scaling', and 'Scheduled Scaling'. The 'Savings' tab is active, displaying a summary of spot usage and savings over the last 3 days. The summary shows 2 Spot Instances, 84 vCPU-hours, 42 Mem(GiB)-hours, and a total On-Demand cost of \$0.47. The total Spot cost is \$0.14, resulting in 70% savings. The average cost per vCPU-hour is \$0.0017, and the average cost per Mem(GiB)-hour is \$0.0034.

Request Id	Request type	Instance Type	State	Capacity	Status	Persistence	Created
sfr-0661ccc5-1890-...	fleet	t3.micro	active	2 of 2	fulfilled	maintain	21 hours a
sir-5dhg73bm	instance	t3.micro	active	i-0e9092dda144...	fulfilled	persistent	21 hours a
sir-54q85zmp	instance	t3.micro	active	i-0f0c7d6a61d6...	fulfilled	persistent	21 hours a

Request Id: sfr-0661ccc5-1890-4d56-9f5a-e673b388e10c

Description Instances History **Savings** Auto Scaling Scheduled Scaling

Your Spot usage and savings details for this Spot Fleet request.
For a high-level savings summary across all of your running Spot Instances, check your Savings Summary.
For detailed reporting on your account-level Spot usage, visit Cost Explorer.

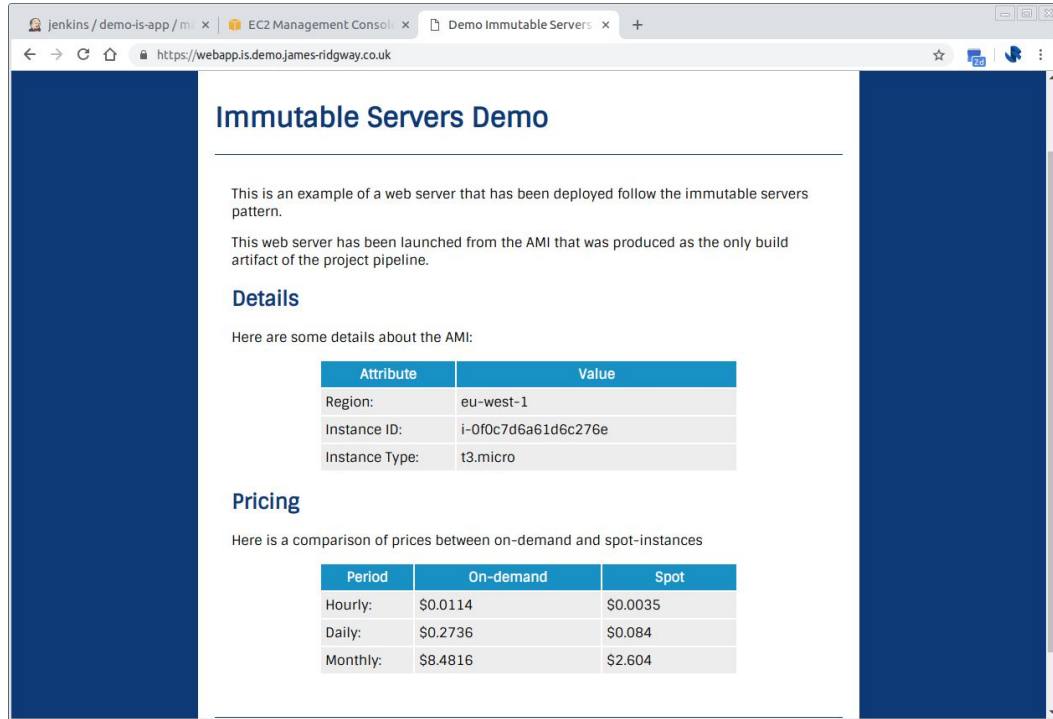
Spot usage and savings Time frame: last 3 days

2	84	42	\$0.47	\$0.14	70%
Spot Instances	vCPU-hours	Mem(GiB)-hours	On-Demand total	Spot total	Savings

Average cost per vCPU-hour: \$0.0017
Average cost per mem(GiB)-hour: \$0.0034

Details

An Application Running on Spot Instances



The screenshot shows a web browser window with the following content:

- Browser tabs: jenkins / demo-is-app / m..., EC2 Management Console, Demo Immutable Servers
- Address bar: https://webapp.is.demojames-ridgway.co.uk
- Page title: Immutable Servers Demo
- Text: "This is an example of a web server that has been deployed follow the immutable servers pattern." and "This web server has been launched from the AMI that was produced as the only build artifact of the project pipeline."
- Section: **Details**
- Text: "Here are some details about the AMI:"
- Table with 2 columns: Attribute, Value
- Table with 3 columns: Period, On-demand, Spot

Attribute	Value
Region:	eu-west-1
Instance ID:	i-0f0c7d6a61d6c276e
Instance Type:	t3.micro

Period	On-demand	Spot
Hourly:	\$0.0114	\$0.0035
Daily:	\$0.2736	\$0.084
Monthly:	\$8.4816	\$2.604

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