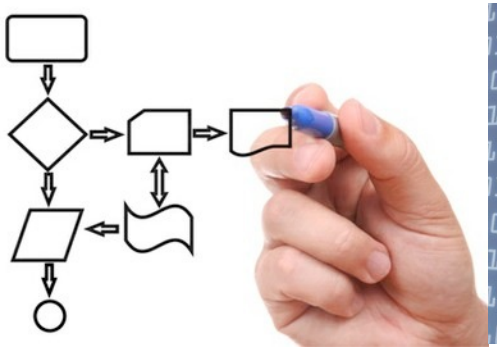
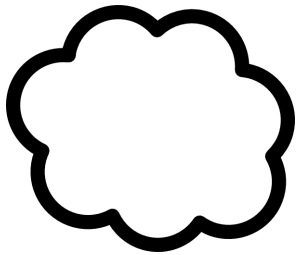
# San Jose State University

CMPE 281

Cloud Technologies

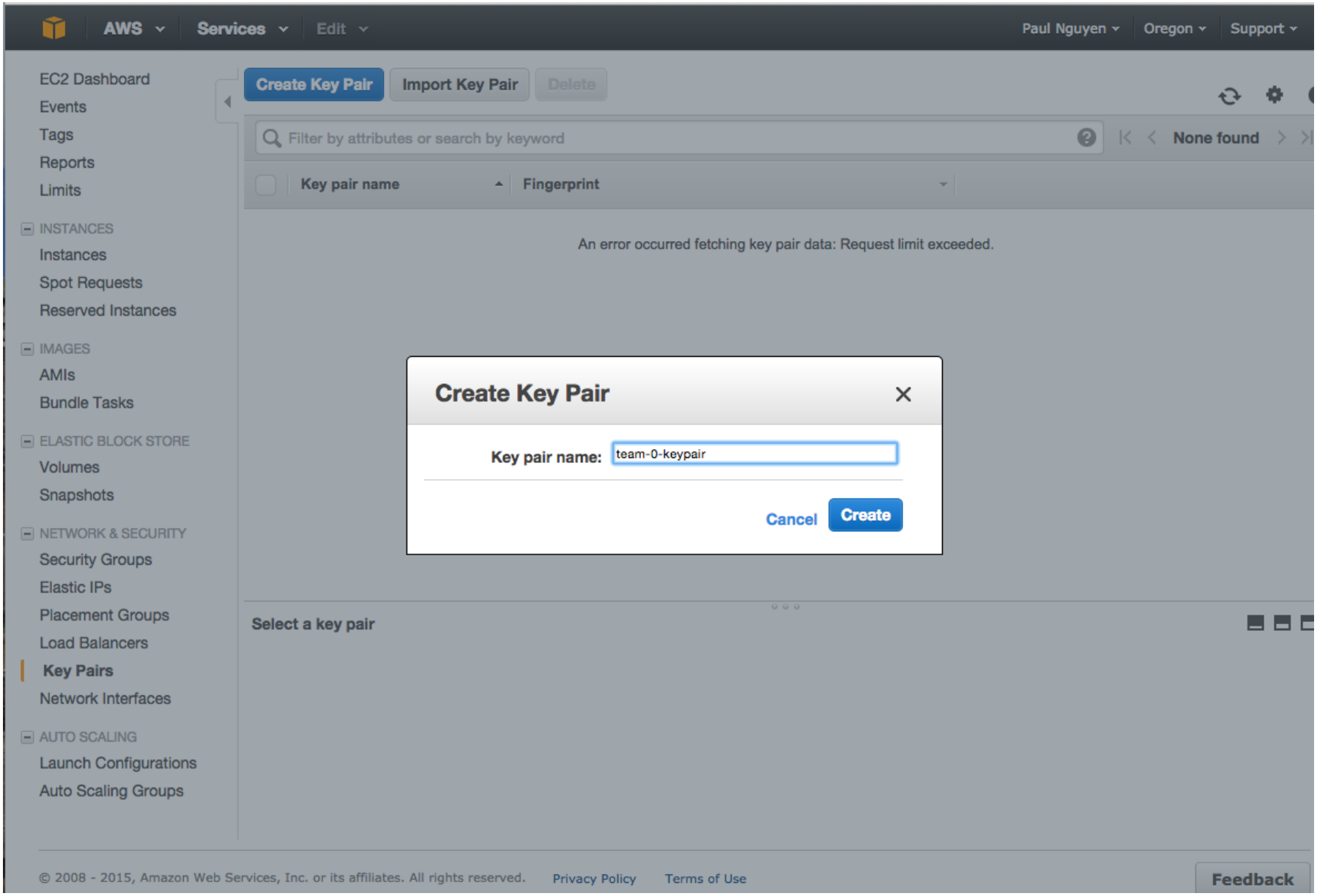
AWS & Grails Lab





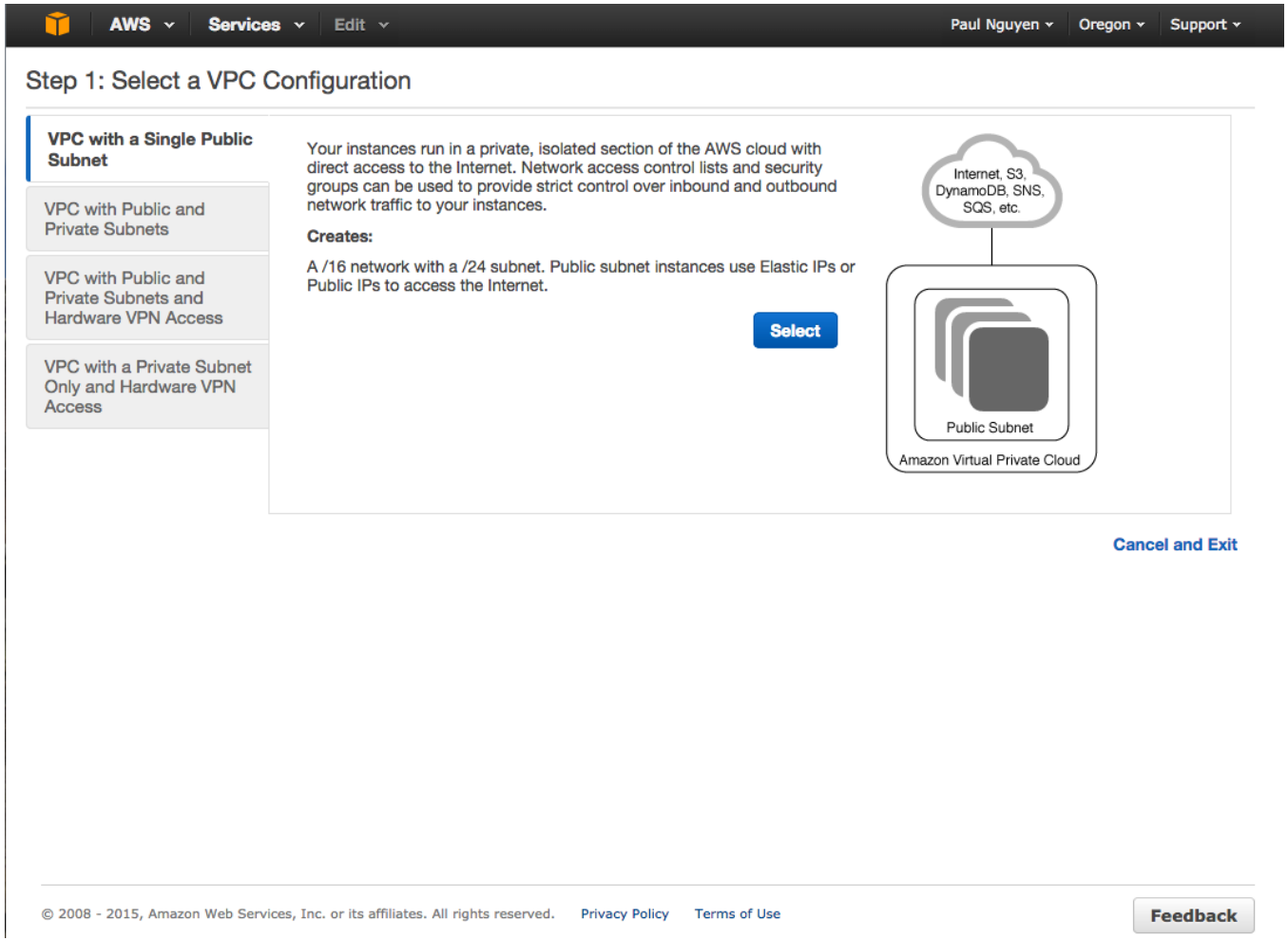
# Setup

# Step #0. Create Team Key Pairs



# Step #1. Setup VPC

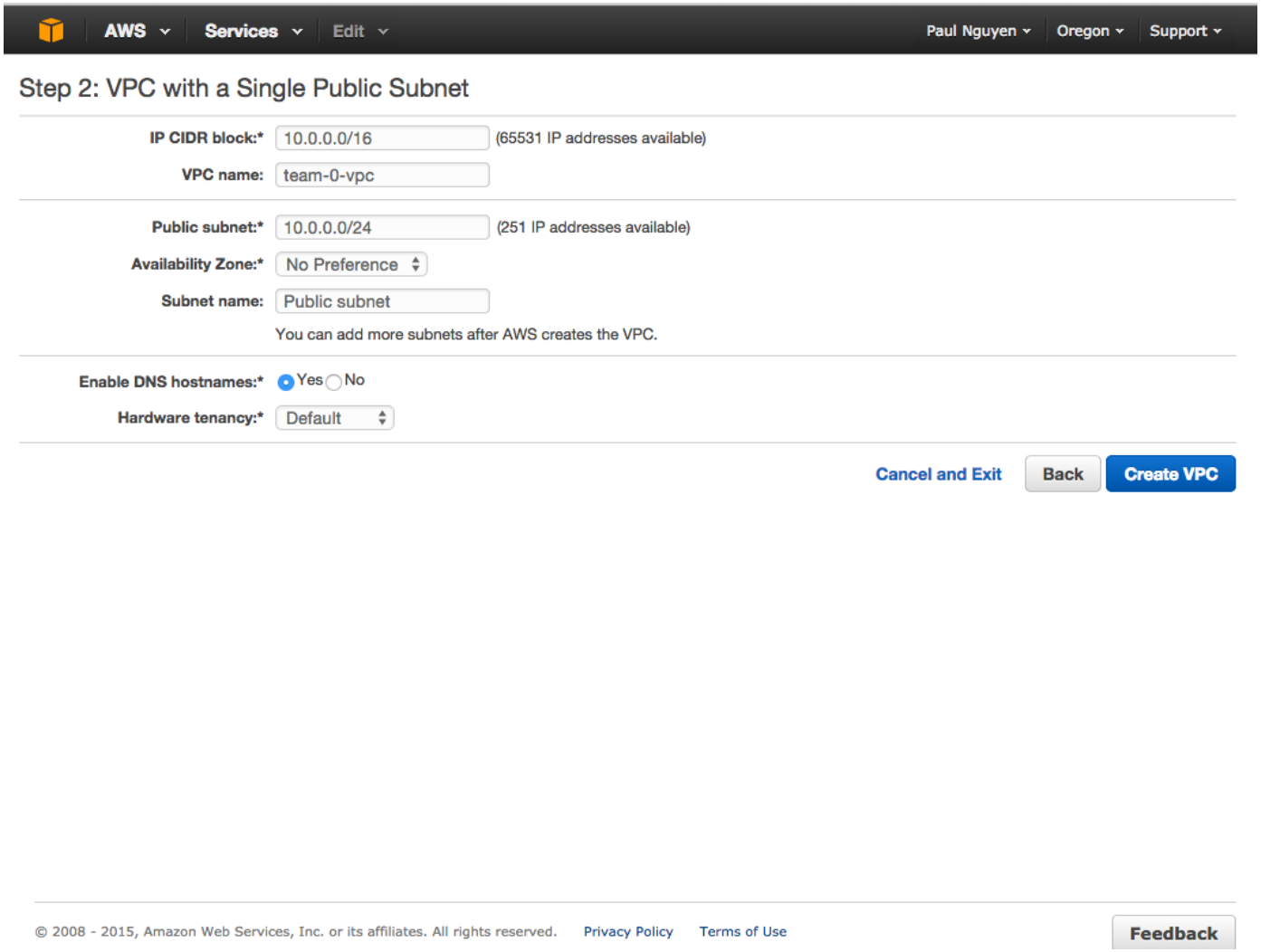
VPC with a Single Public Subnet,




http://docs.aws.amazon.com/AmazonVPC/latest/GettingStartedGuide/ExerciseOverview.html

# Step #1. Setup VPC

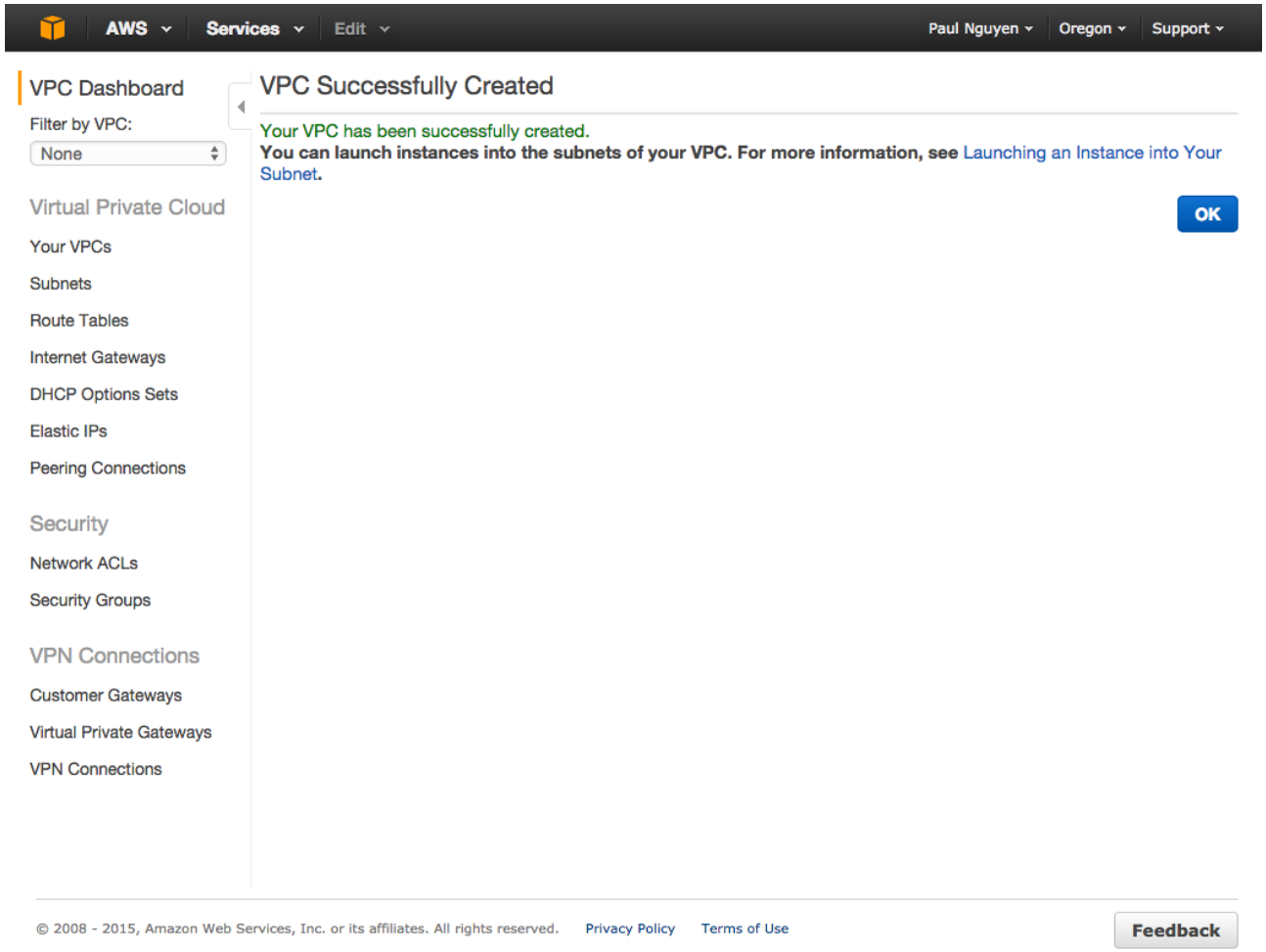
VPC with a Single Public Subnet,




http://docs.aws.amazon.com/AmazonVPC/latest/GettingStartedGuide/ExerciseOverview.html

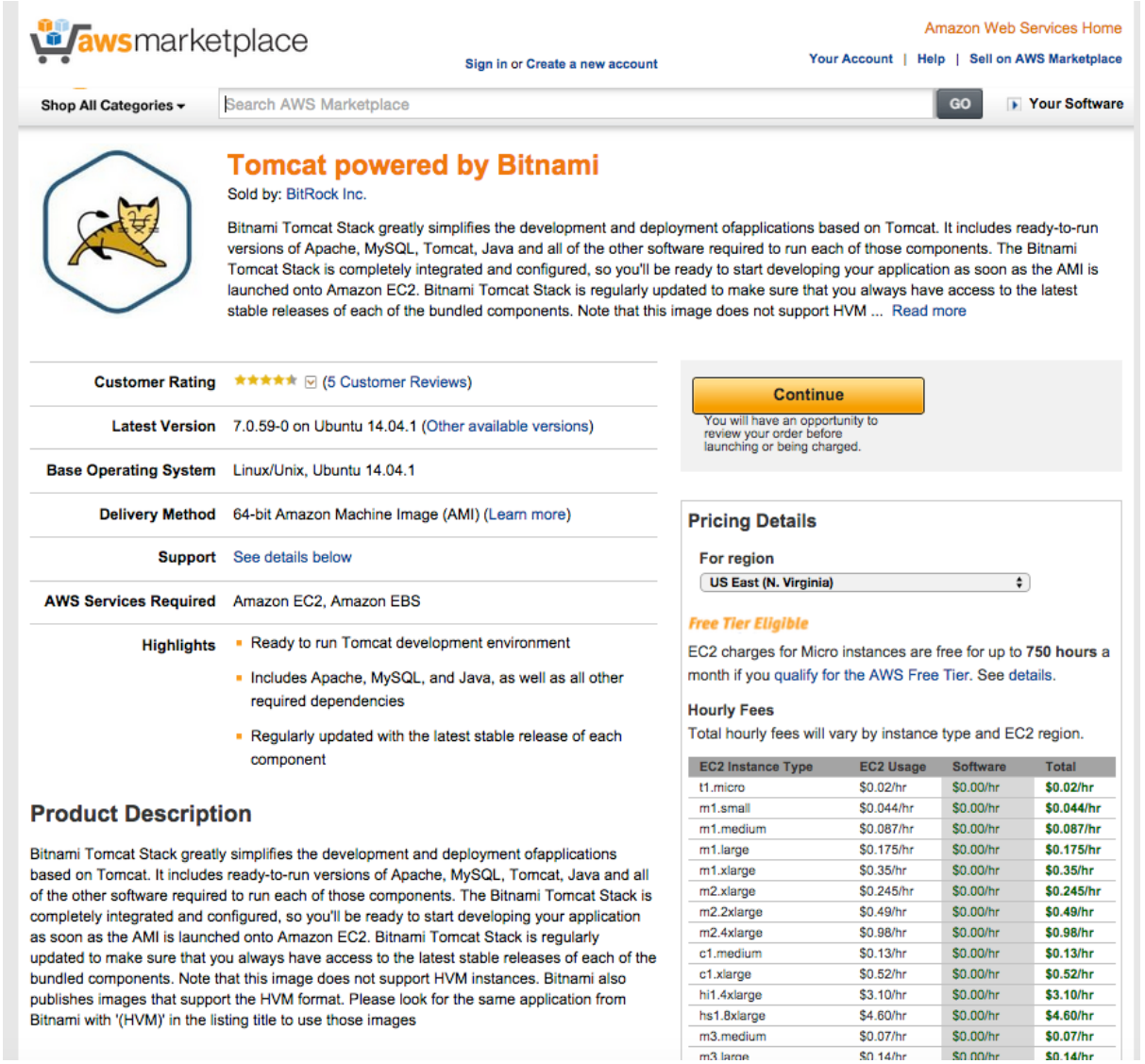
# Step #1. Setup VPC

VPC with a Single Public Subnet,

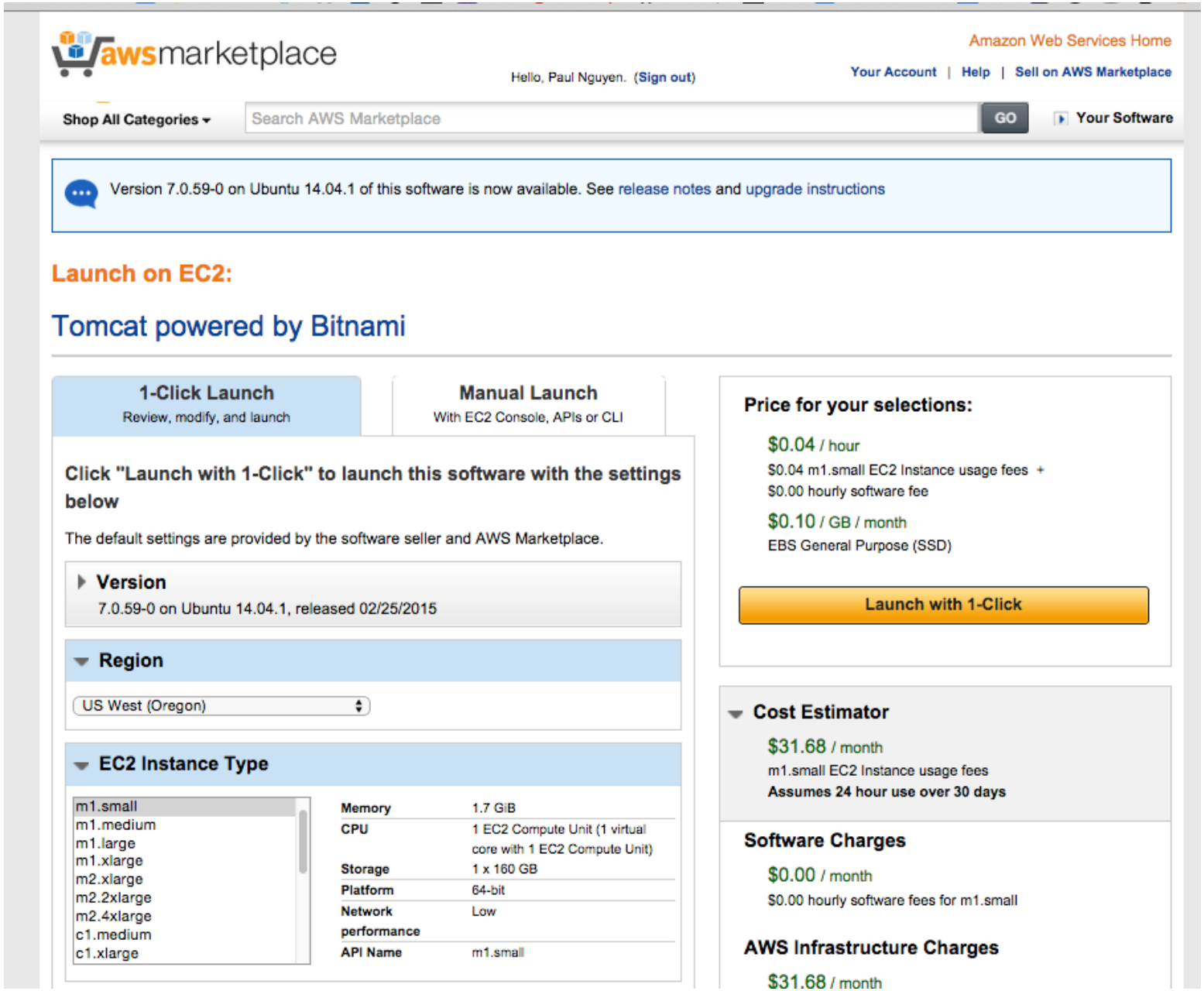



http://docs.aws.amazon.com/AmazonVPC/latest/GettingStartedGuide/ExerciseOverview.html

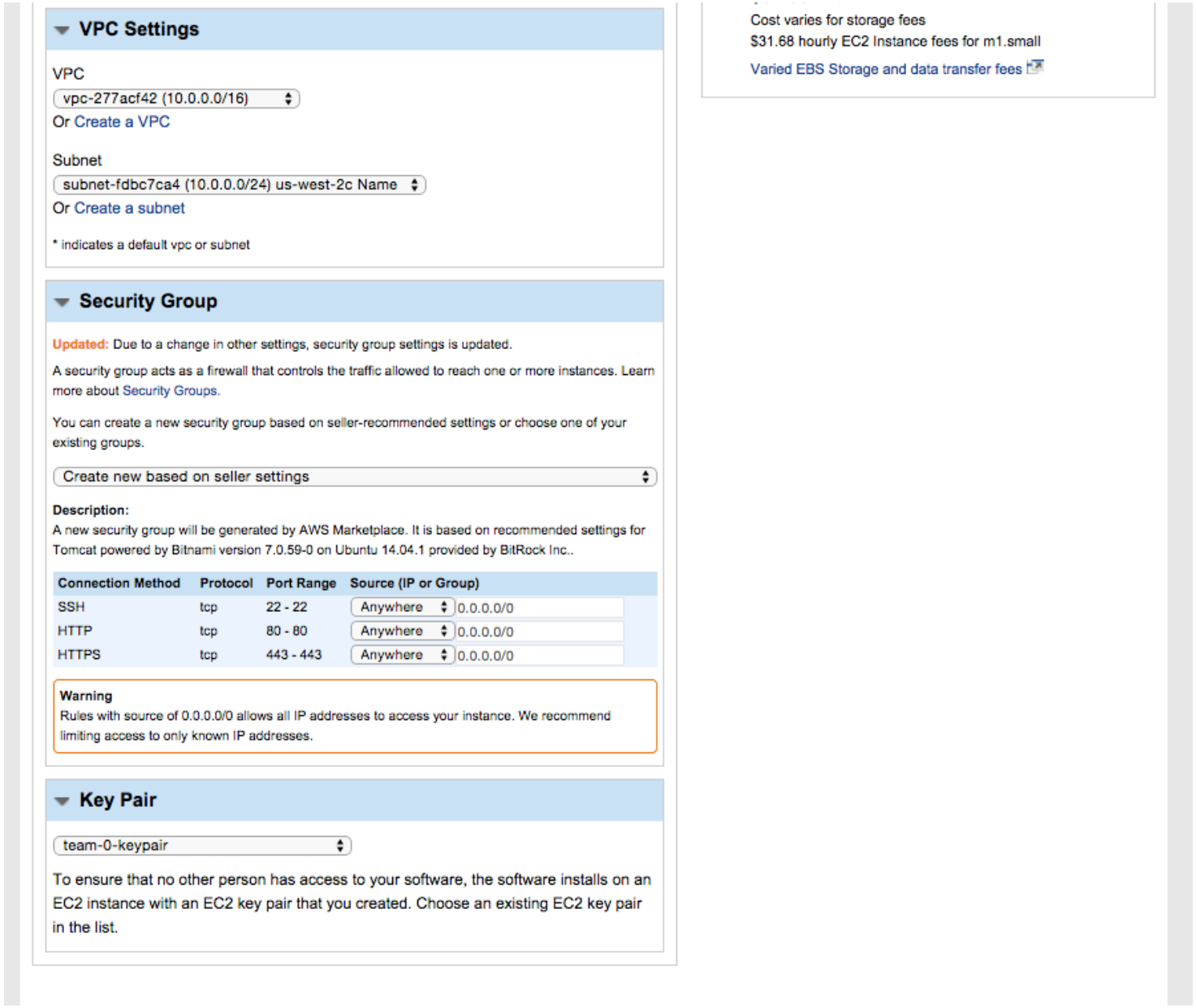
# Step #2. Deploy Tomcat VM



# Step #2. Deploy Tomcat VM



# Step #2. Deploy Tomcat VM



### VPC Settings

VPC

vpc-277acf42 (10.0.0.0/16)

Or Create a VPC

Subnet

subnet-fdbc7ca4 (10.0.0.0/24) us-west-2c Name

Or Create a subnet

* indicates a default vpc or subnet

### Security Group

Updated: Due to a change in other settings, security group settings is updated.

A security group acts as a firewall that controls the traffic allowed to reach one or more instances. Learn more about Security Groups.

You can create a new security group based on seller-recommended settings or choose one of your existing groups.

Create new based on seller settings

**Description:**
A new security group will be generated by AWS Marketplace. It is based on recommended settings for Tomcat powered by Bitnami version 7.0.59-0 on Ubuntu 14.04.1 provided by BitRock Inc..

| Connection Method | Protocol | Port Range | Source (IP or Group) |
|---|---|---|---|
| SSH | tcp | 22 - 22 | Anywhere 0.0.0.0/0 |
| HTTP | tcp | 80 - 80 | Anywhere 0.0.0.0/0 |
| HTTPS | tcp | 443 - 443 | Anywhere 0.0.0.0/0 |

**Warning**
Rules with source of 0.0.0.0/0 allows all IP addresses to access your instance. We recommend limiting access to only known IP addresses.

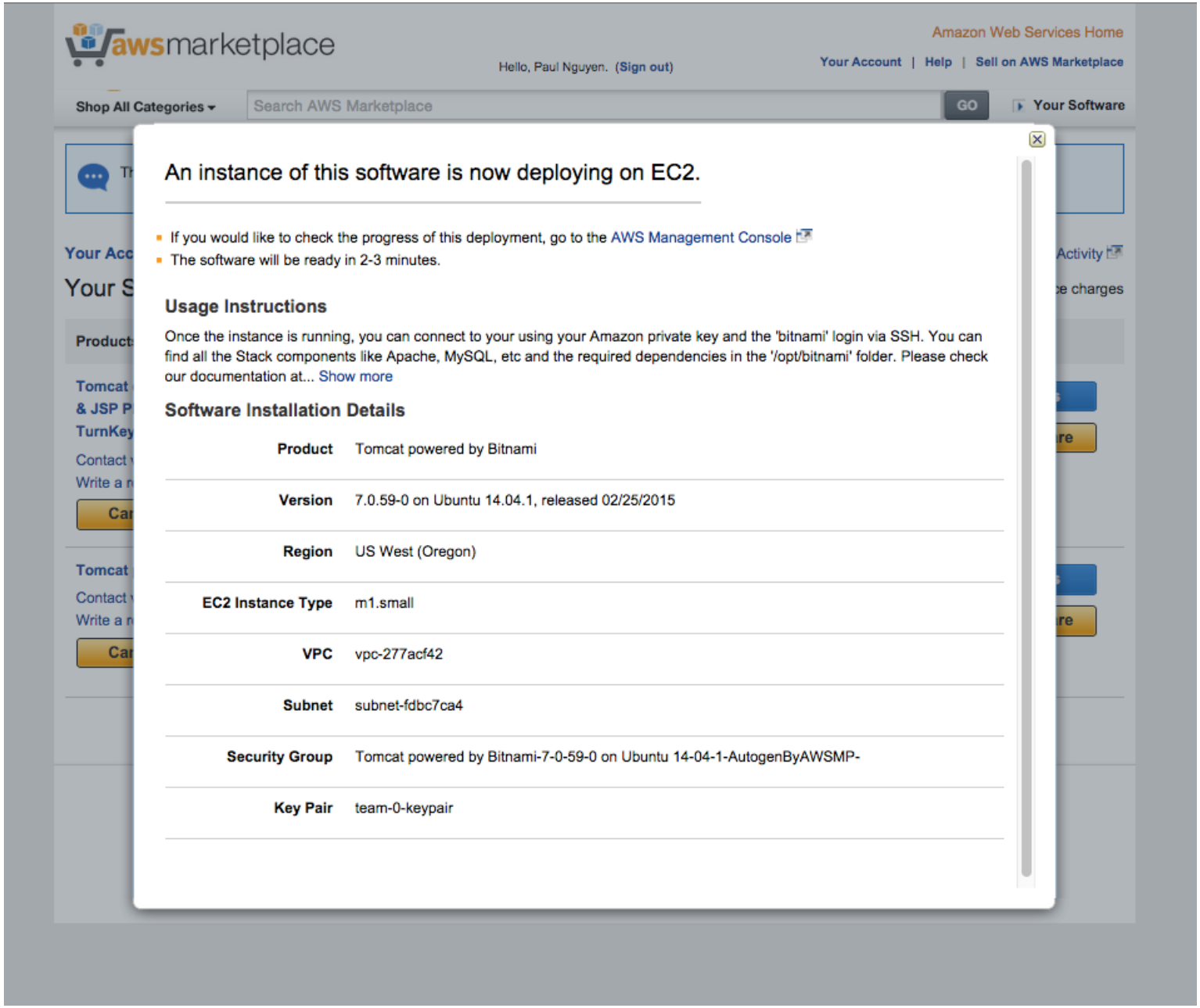
### Key Pair

team-0-keypair

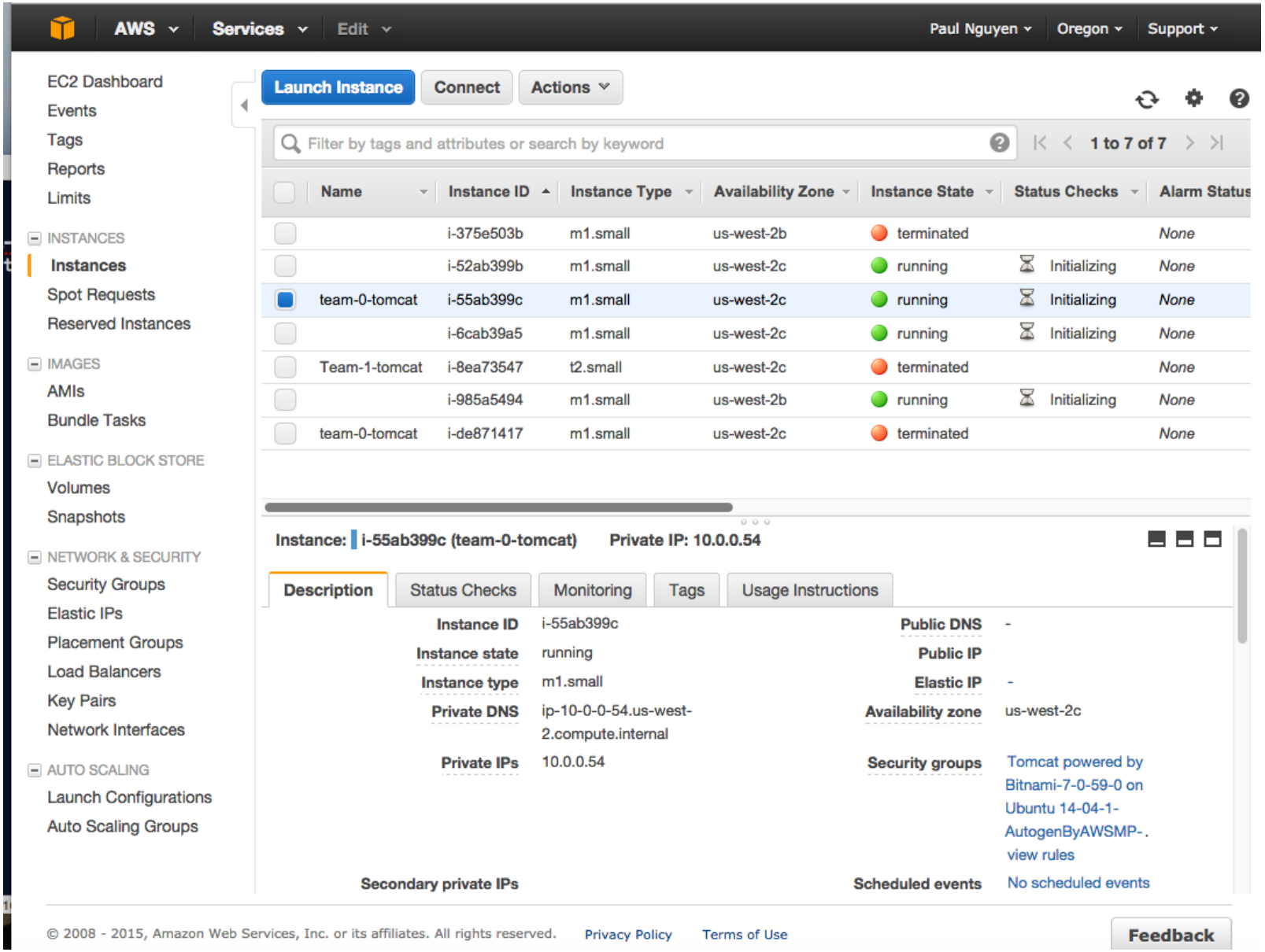
To ensure that no other person has access to your software, the software installs on an EC2 instance with an EC2 key pair that you created. Choose an existing EC2 key pair in the list.

Cost varies for storage fees
$31.68 hourly EC2 Instance fees for m1.small
Varied EBS Storage and data transfer fees

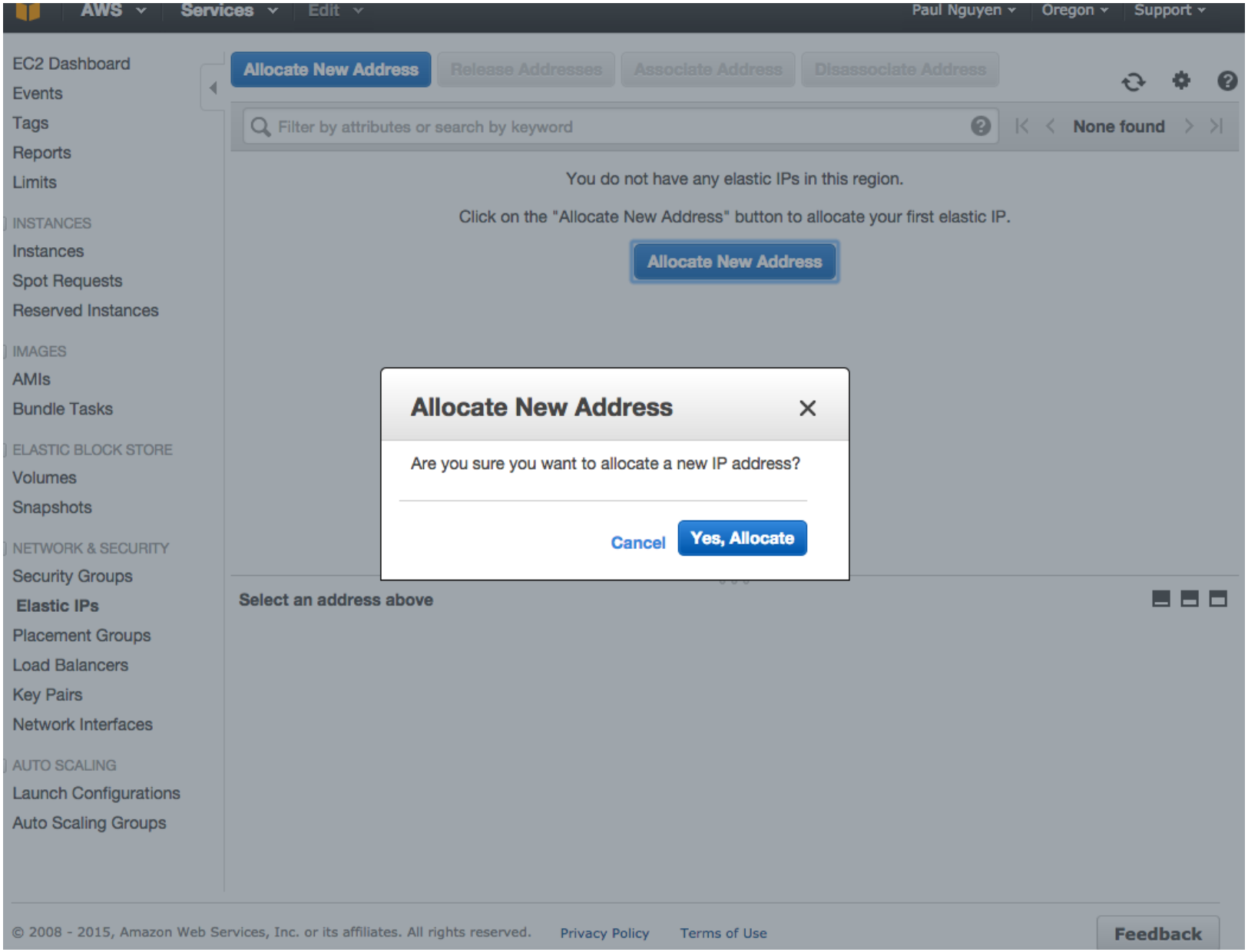
# Step #2. Deploy Tomcat VM



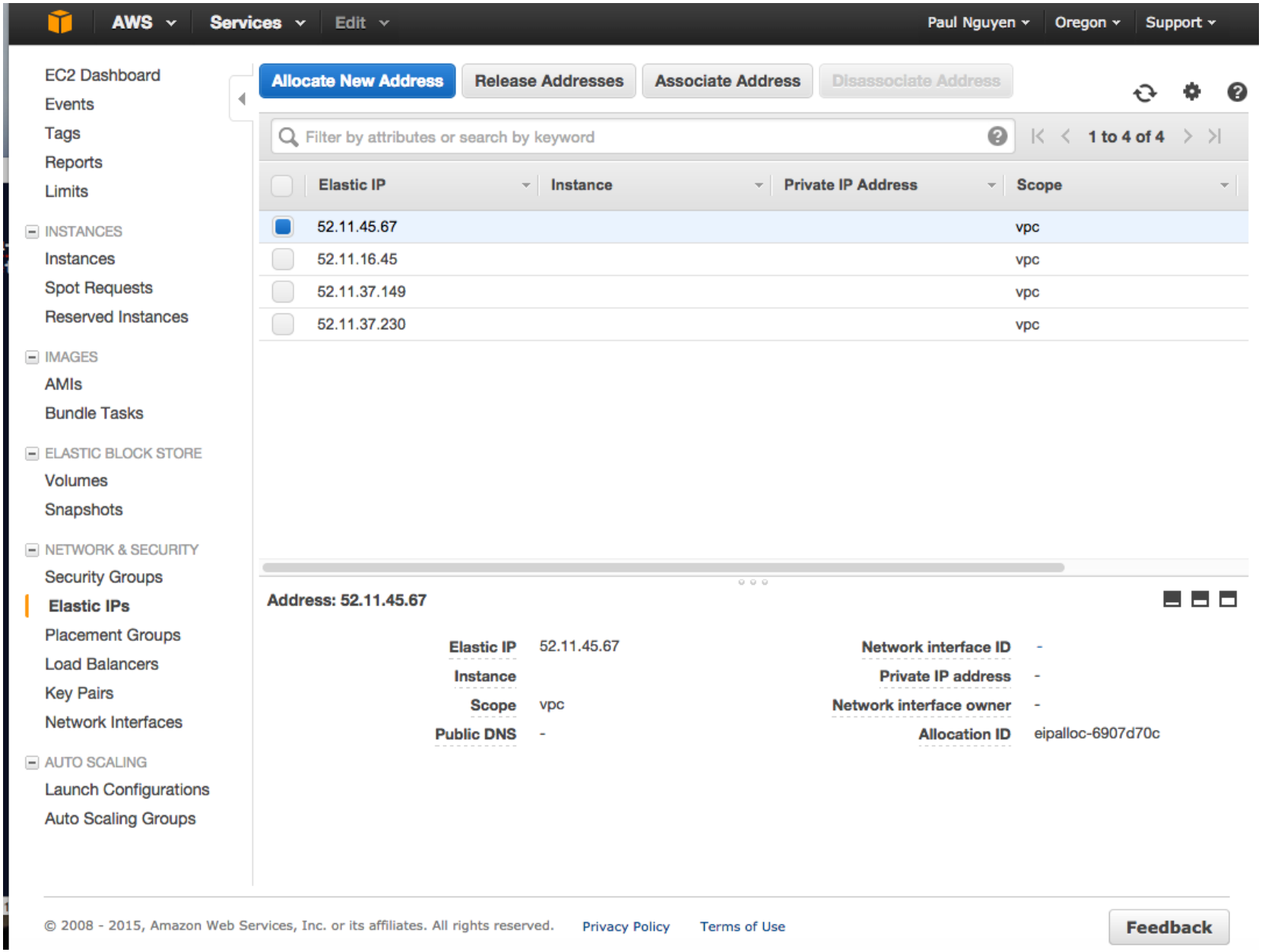
# Step #2. Deploy Tomcat VM



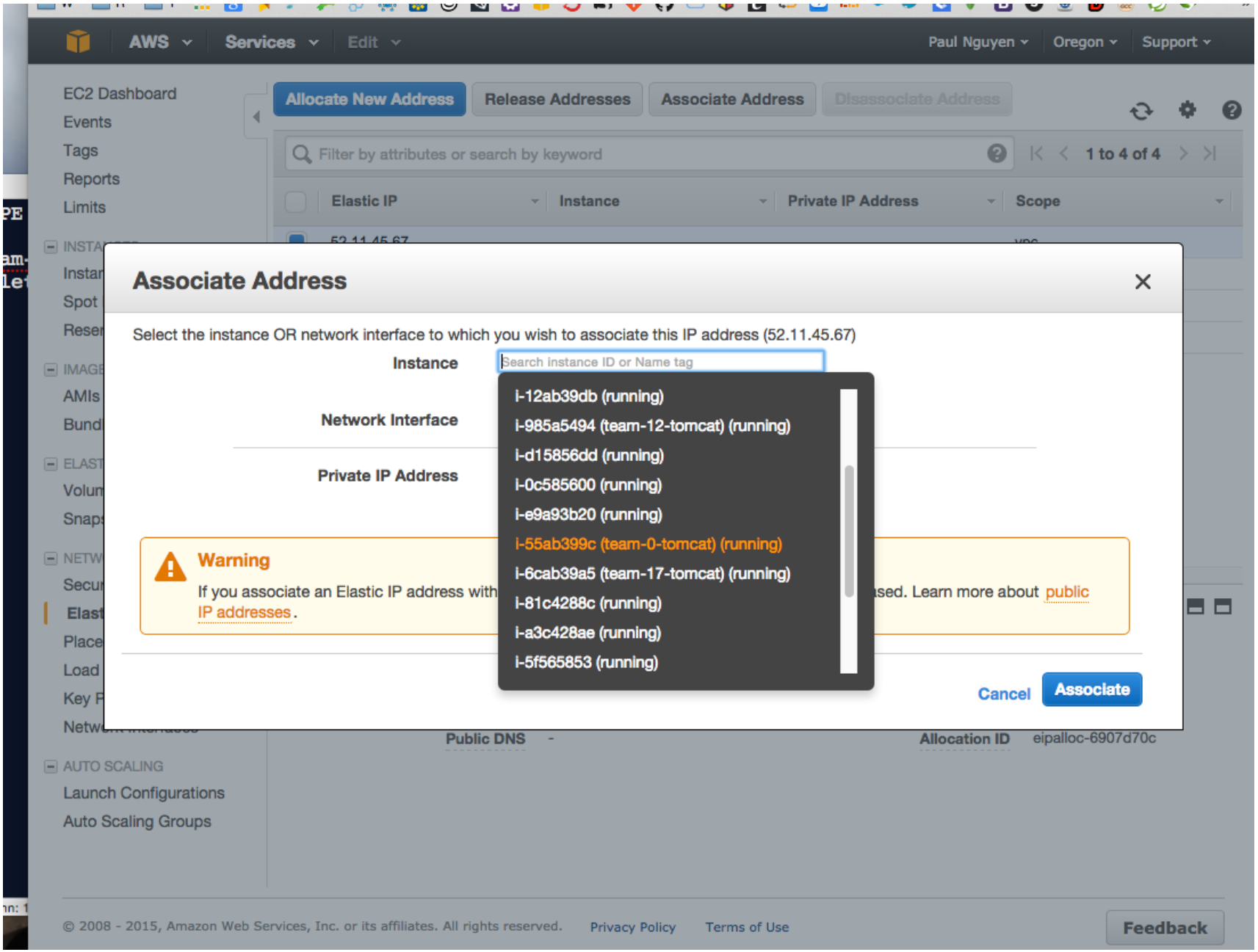
# Step #3. Setup Elastic IP



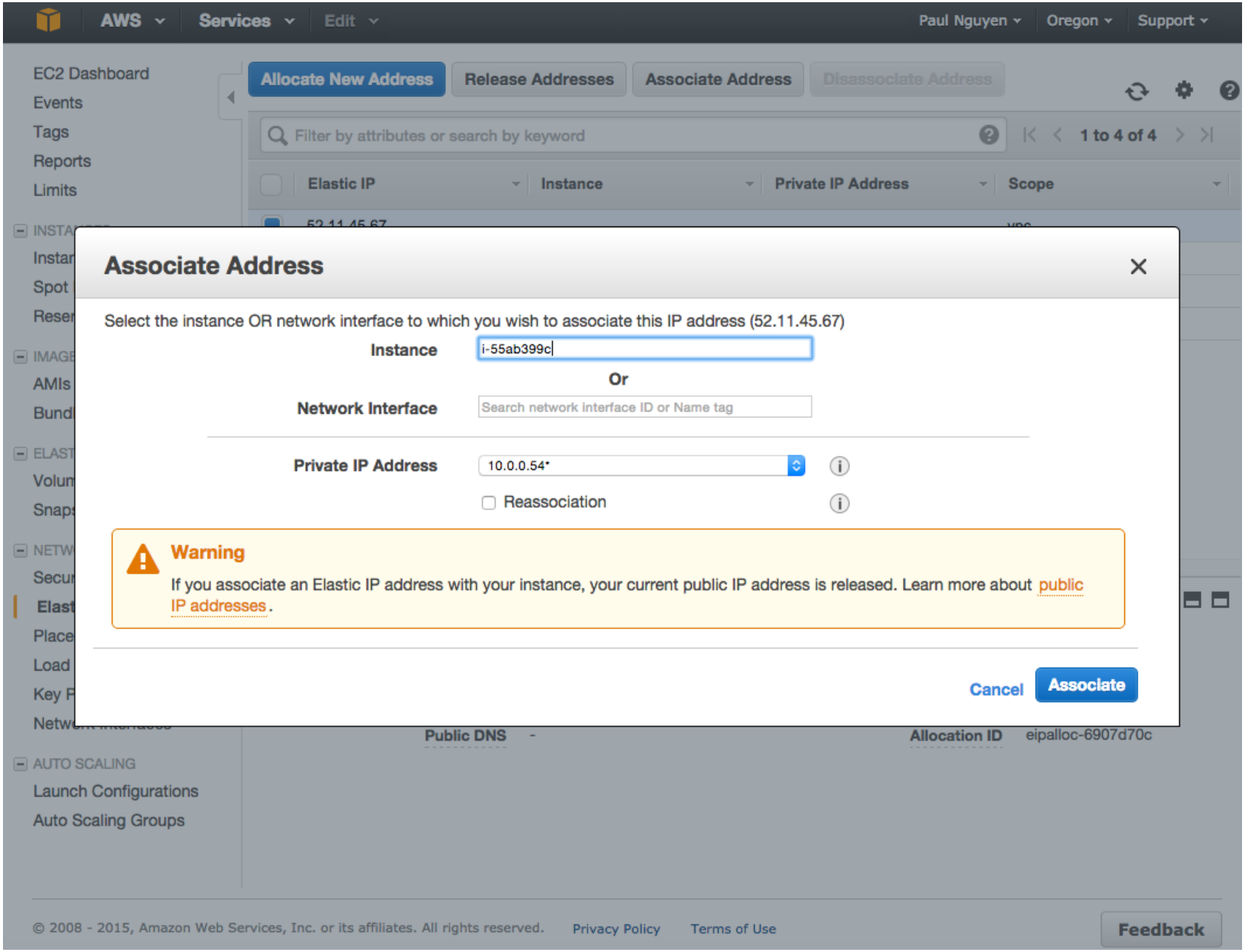
# Step #3. Setup Elastic IP



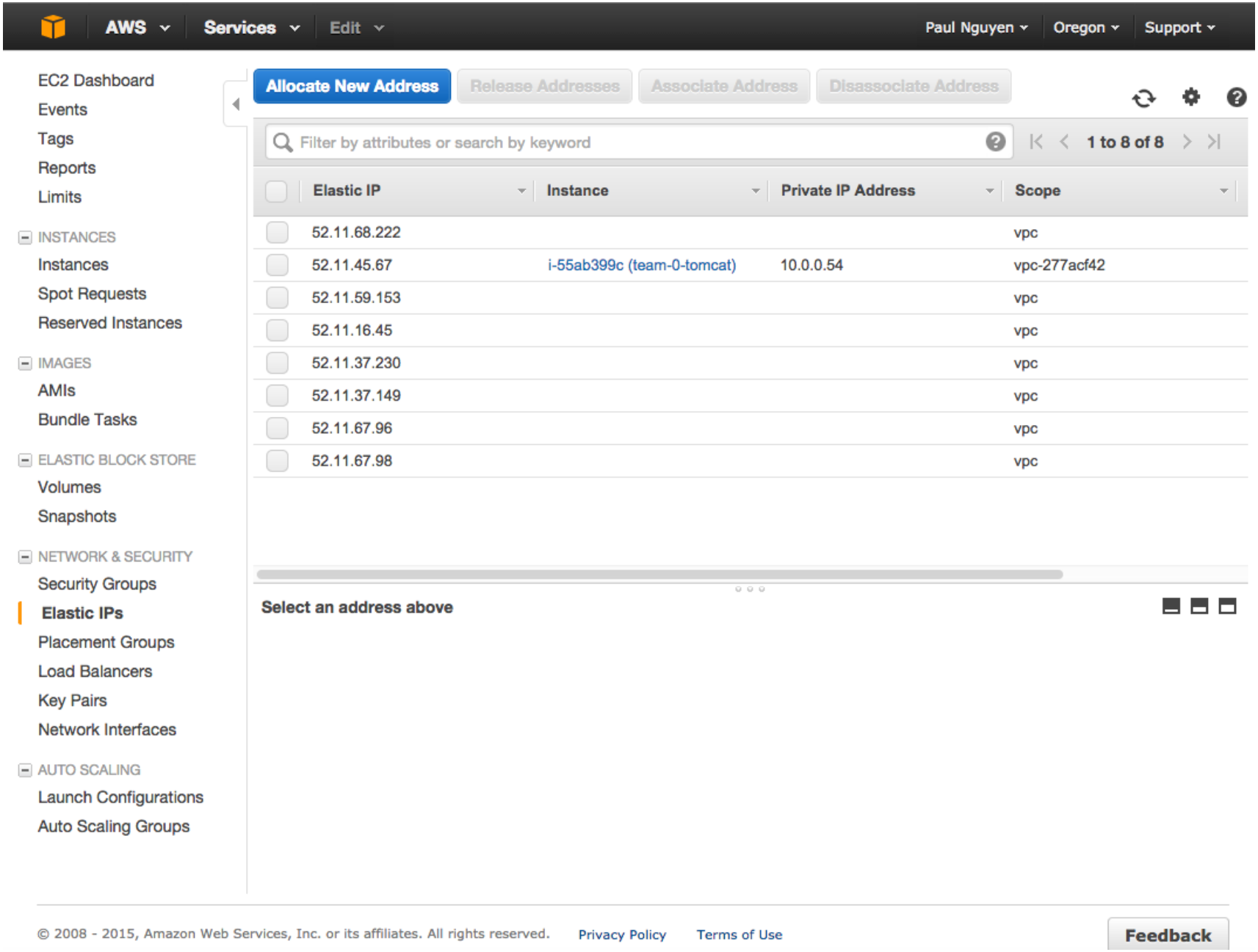
# Step #3. Setup Elastic IP



# Step #3. Setup Elastic IP

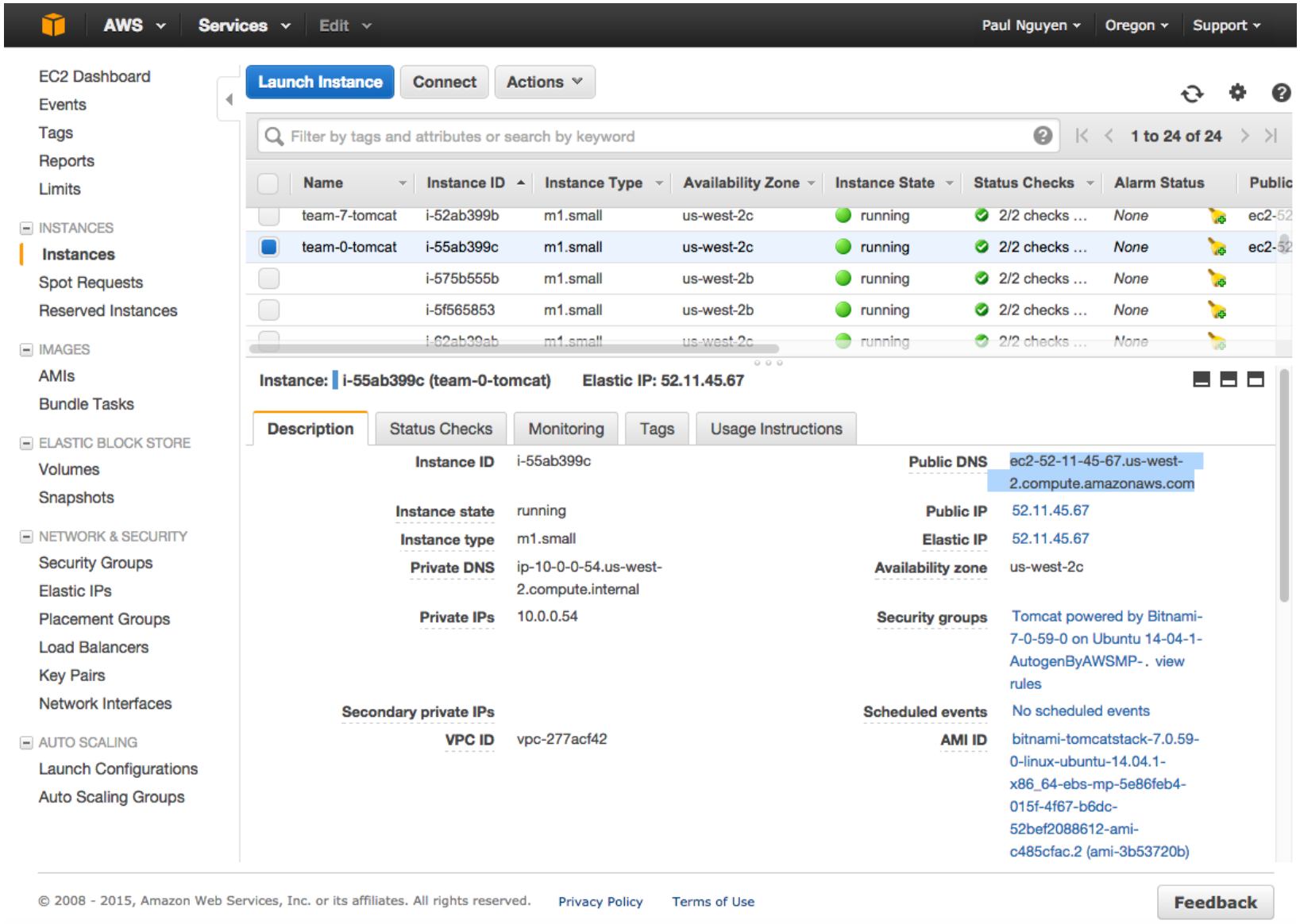


# Step #3. Setup Elastic IP

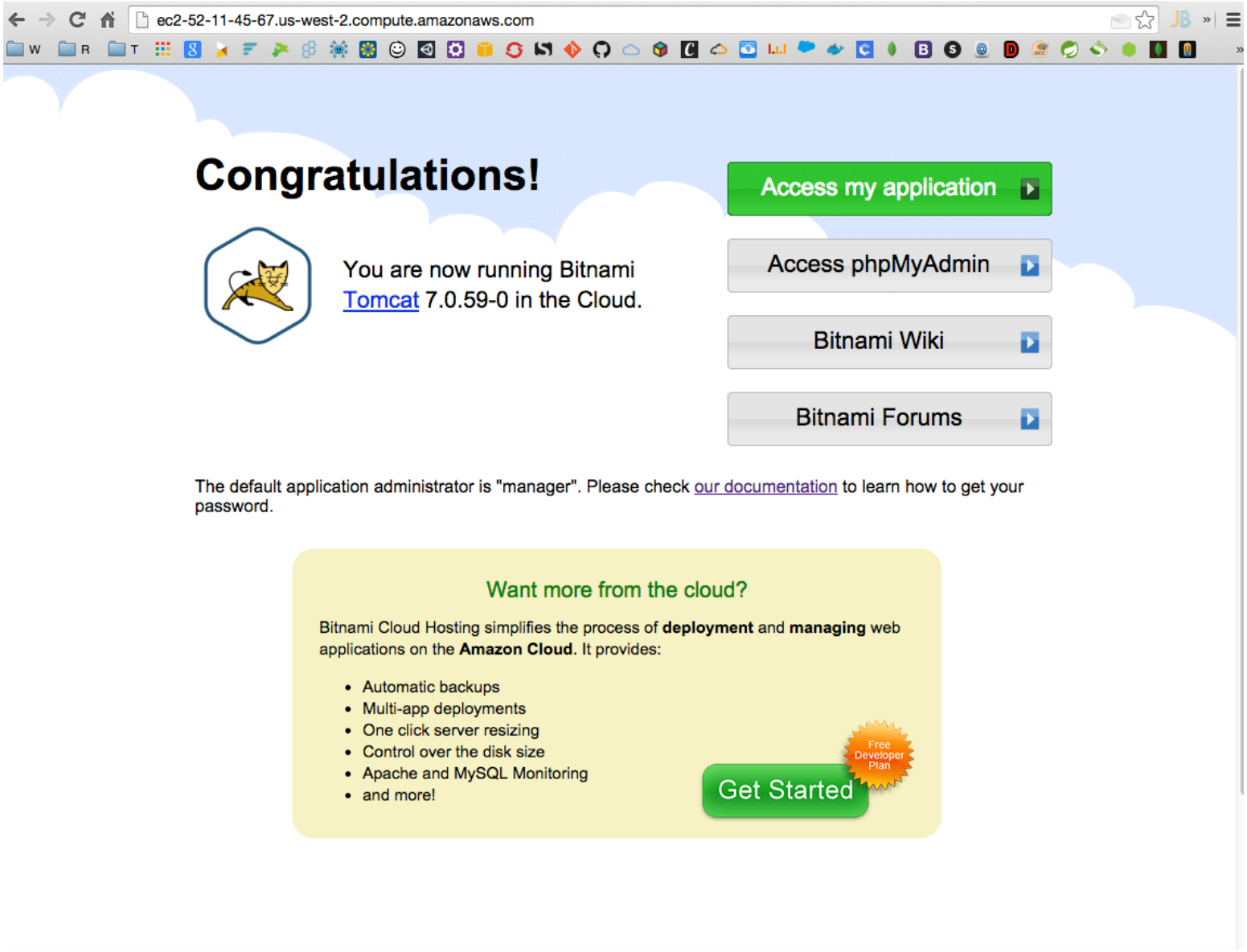


# Step #3. Setup Elastic IP / Test DNS

ec2-52-11-45-67.us-west-2.compute.amazonaws.com

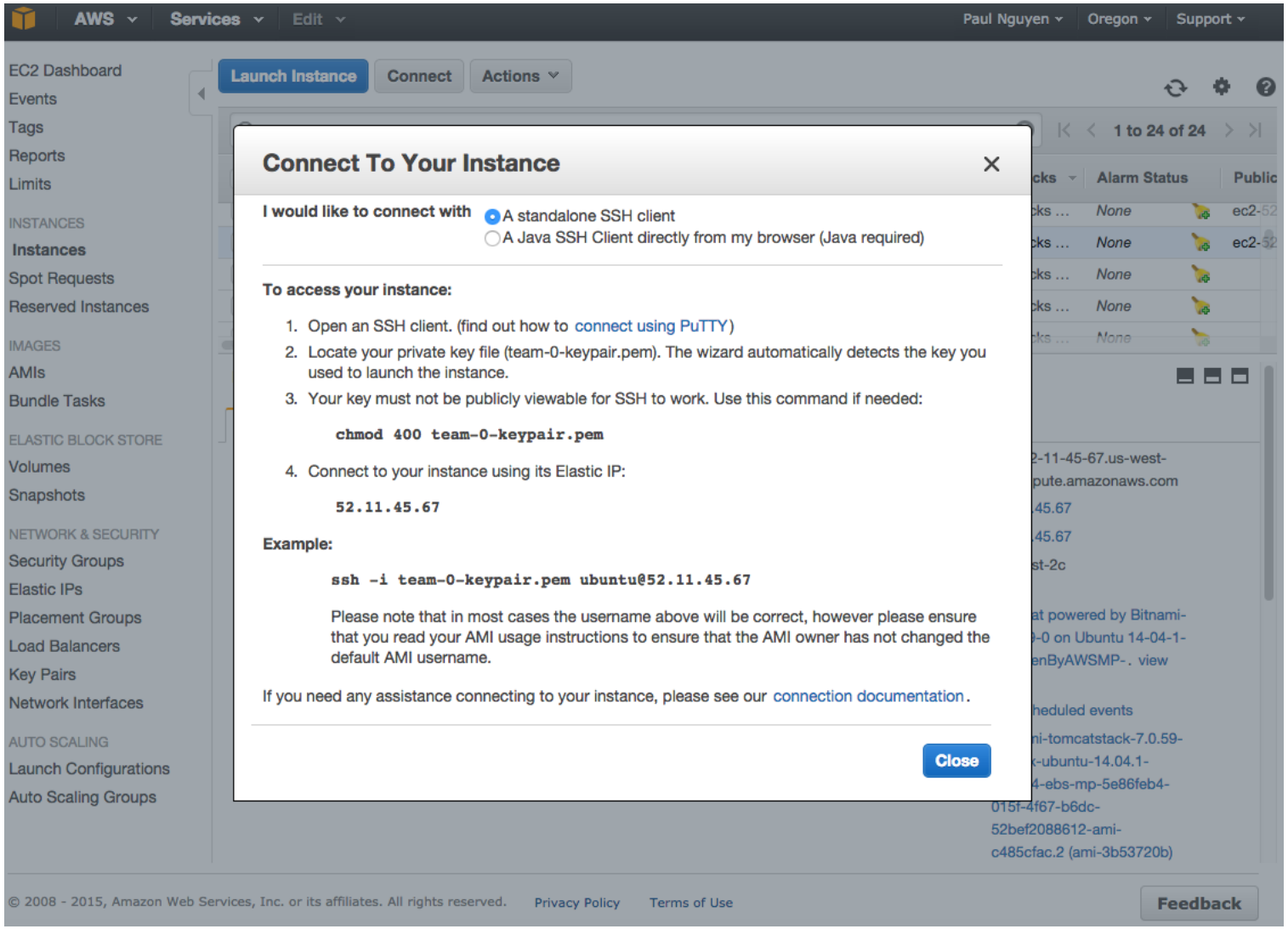


# Step #3. Setup Elastic IP / Test DNS



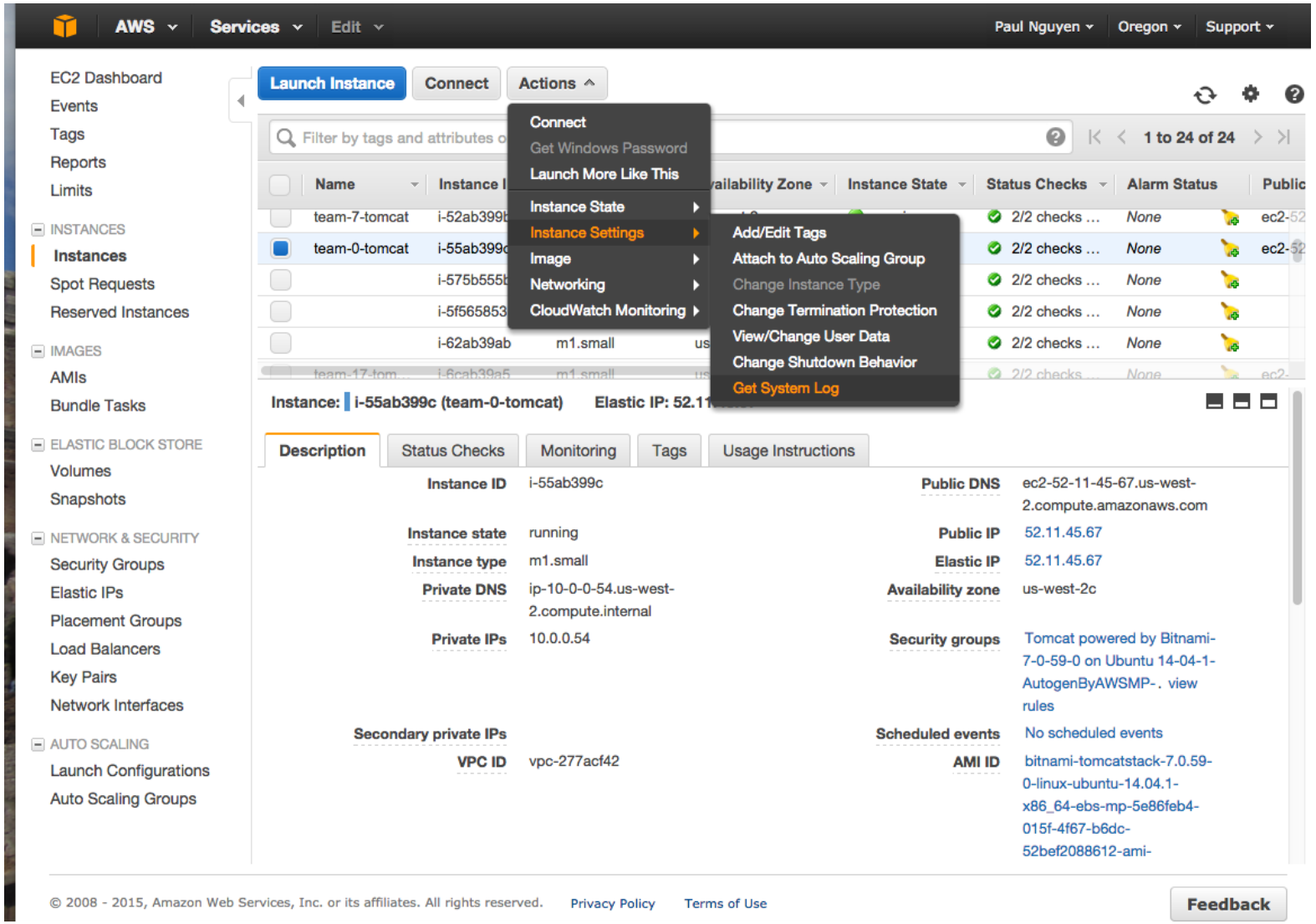
# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com



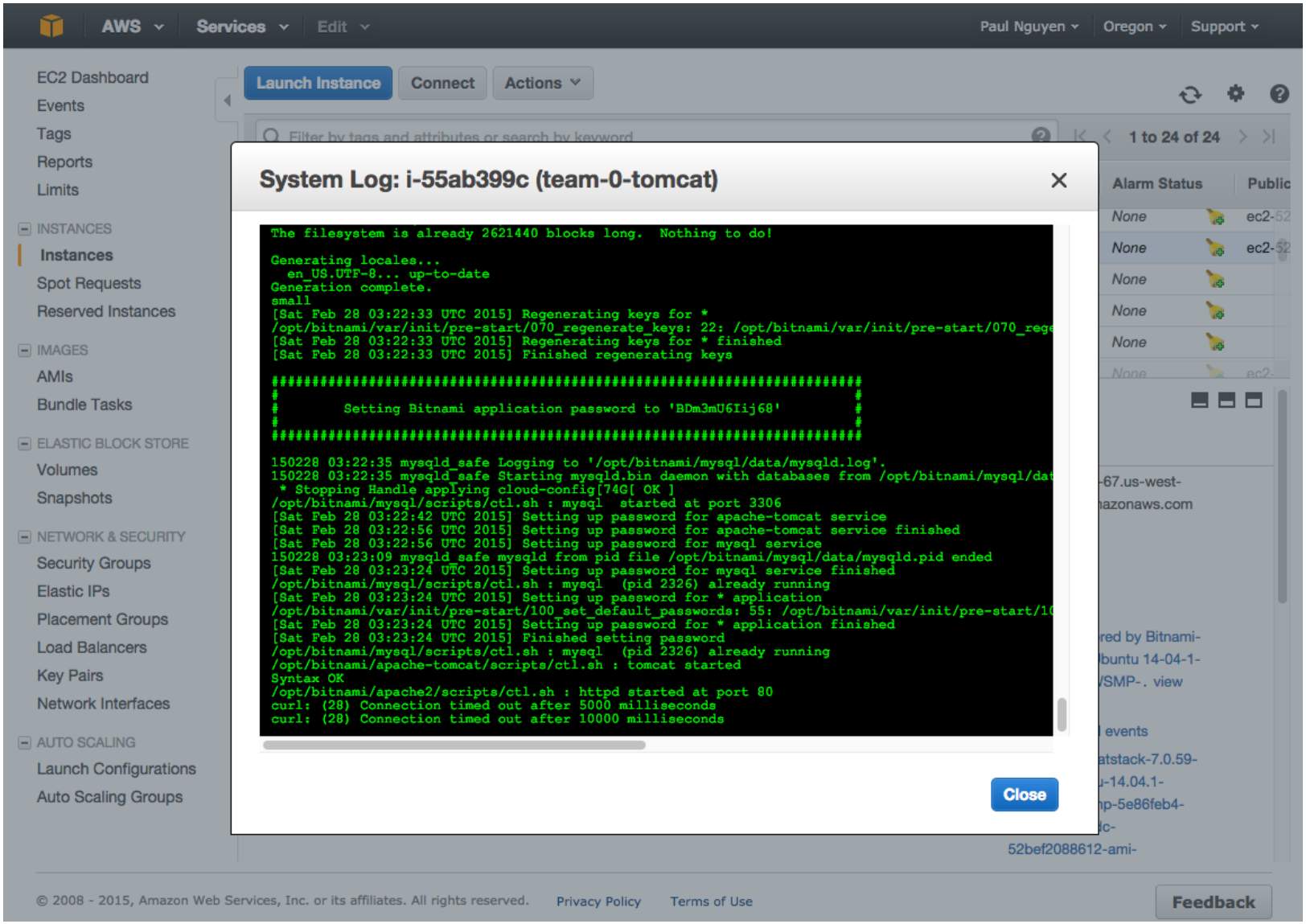
# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com



# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com



# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com

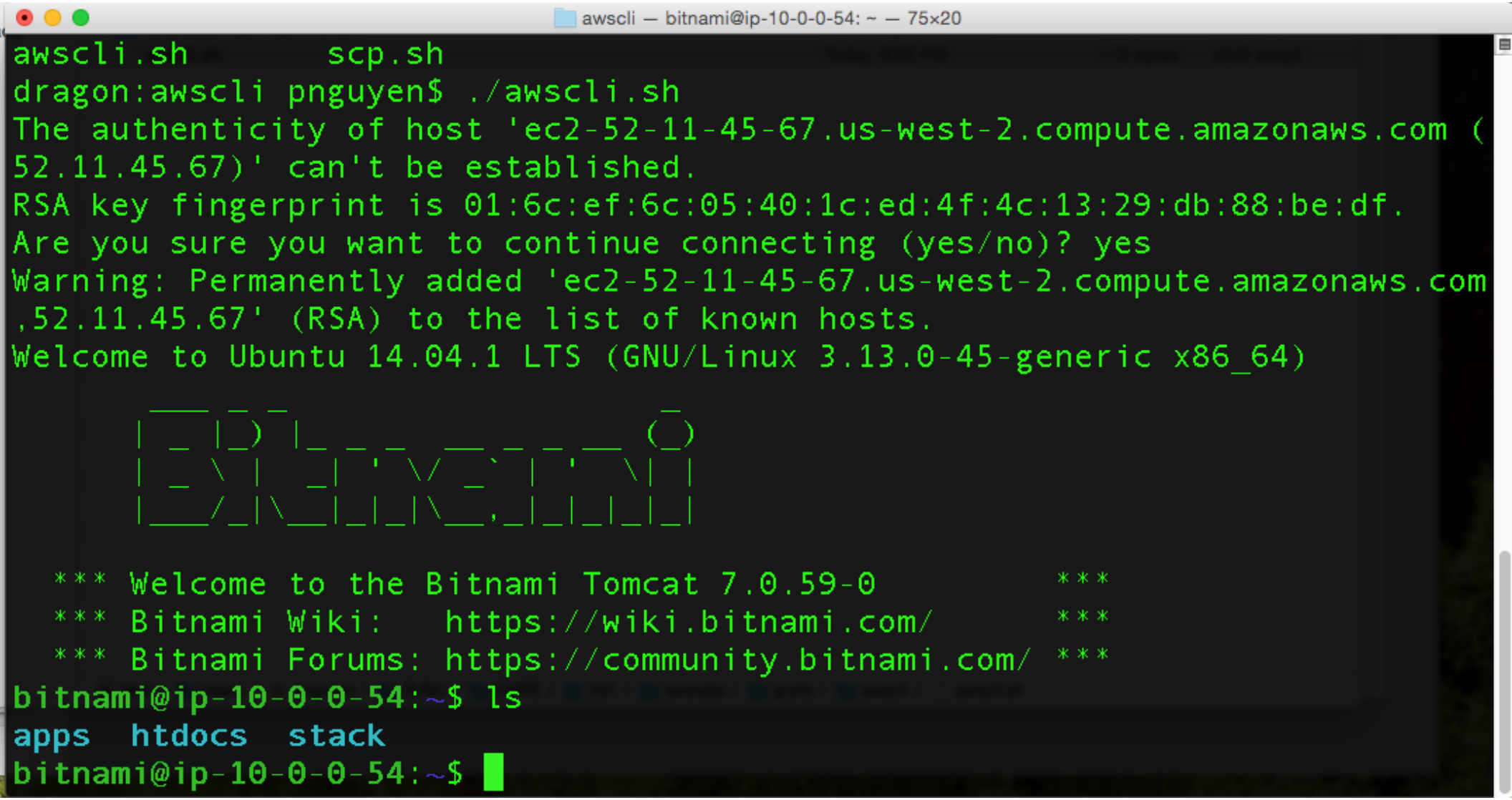
awscli.sh

```
1  # bitnami gen password: BDm3mU6Iij68
2  ssh -i team-0-keypair.pem ubuntu@ec2-52-11-45-67.us-west-2.compute.amazonaws.com
3
```

Line: 2 Column: 81 Shell Script (Bash) Tab Size: 4

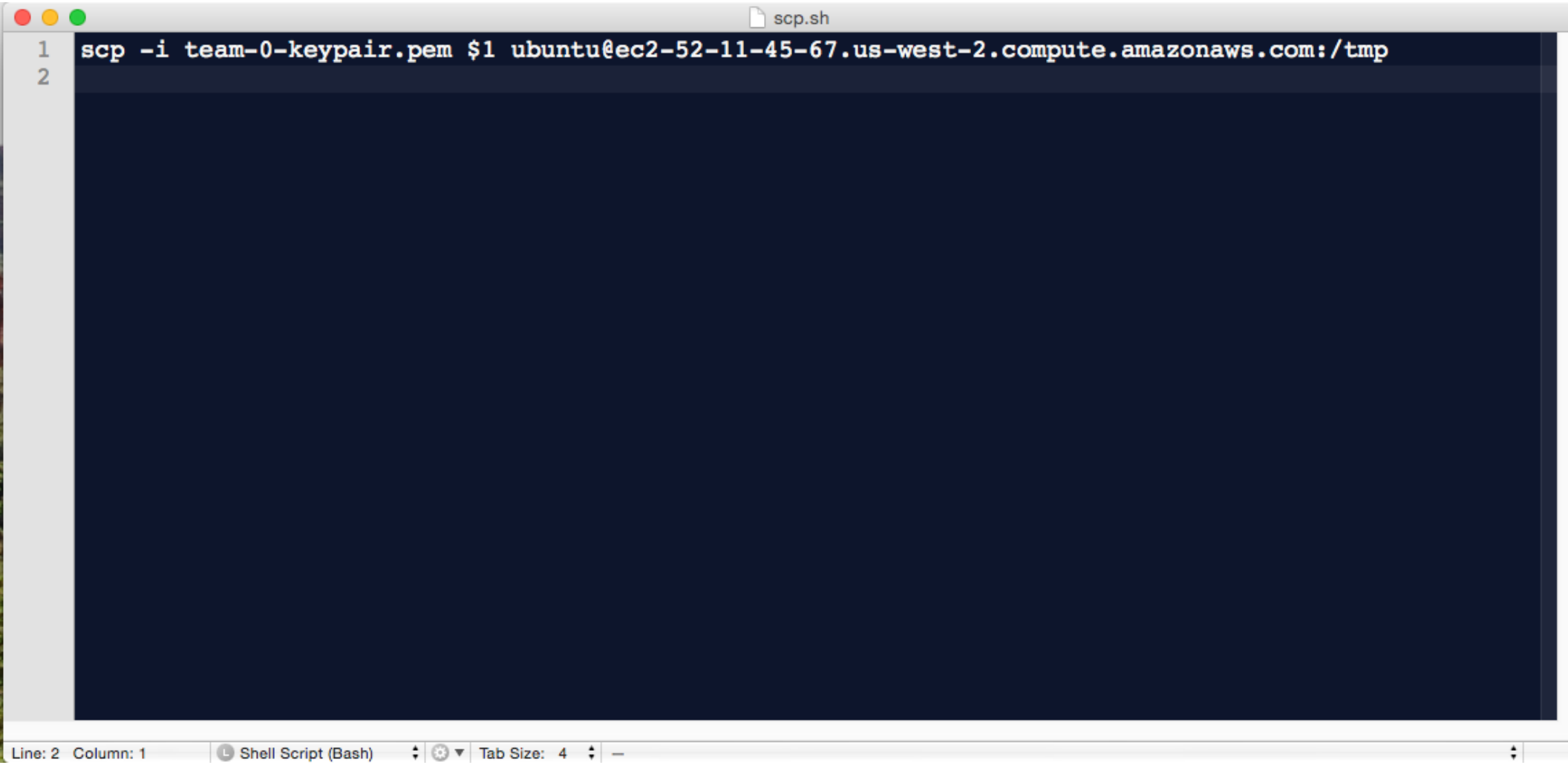
# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com



# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com

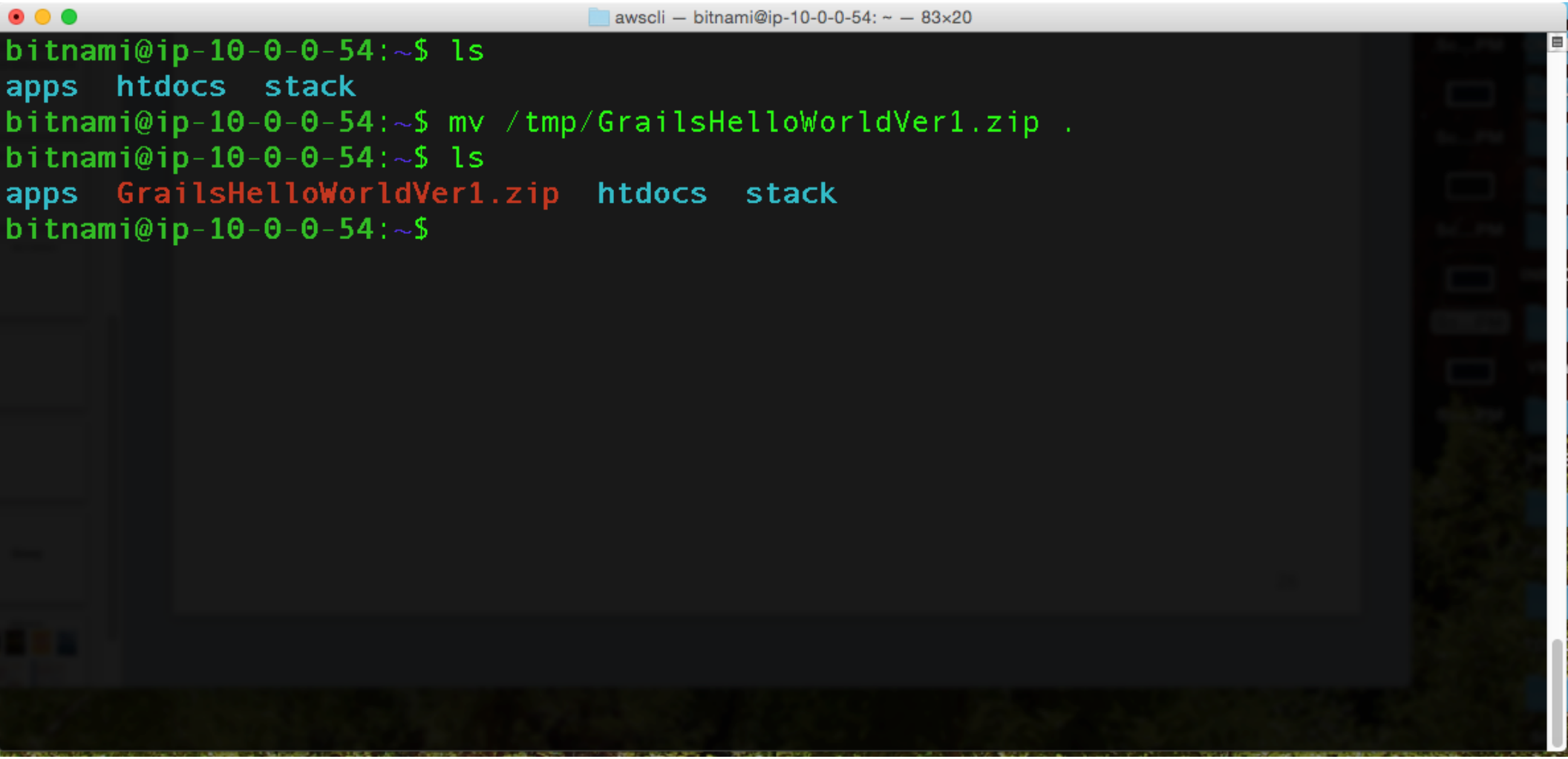


# Step #4.  SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com

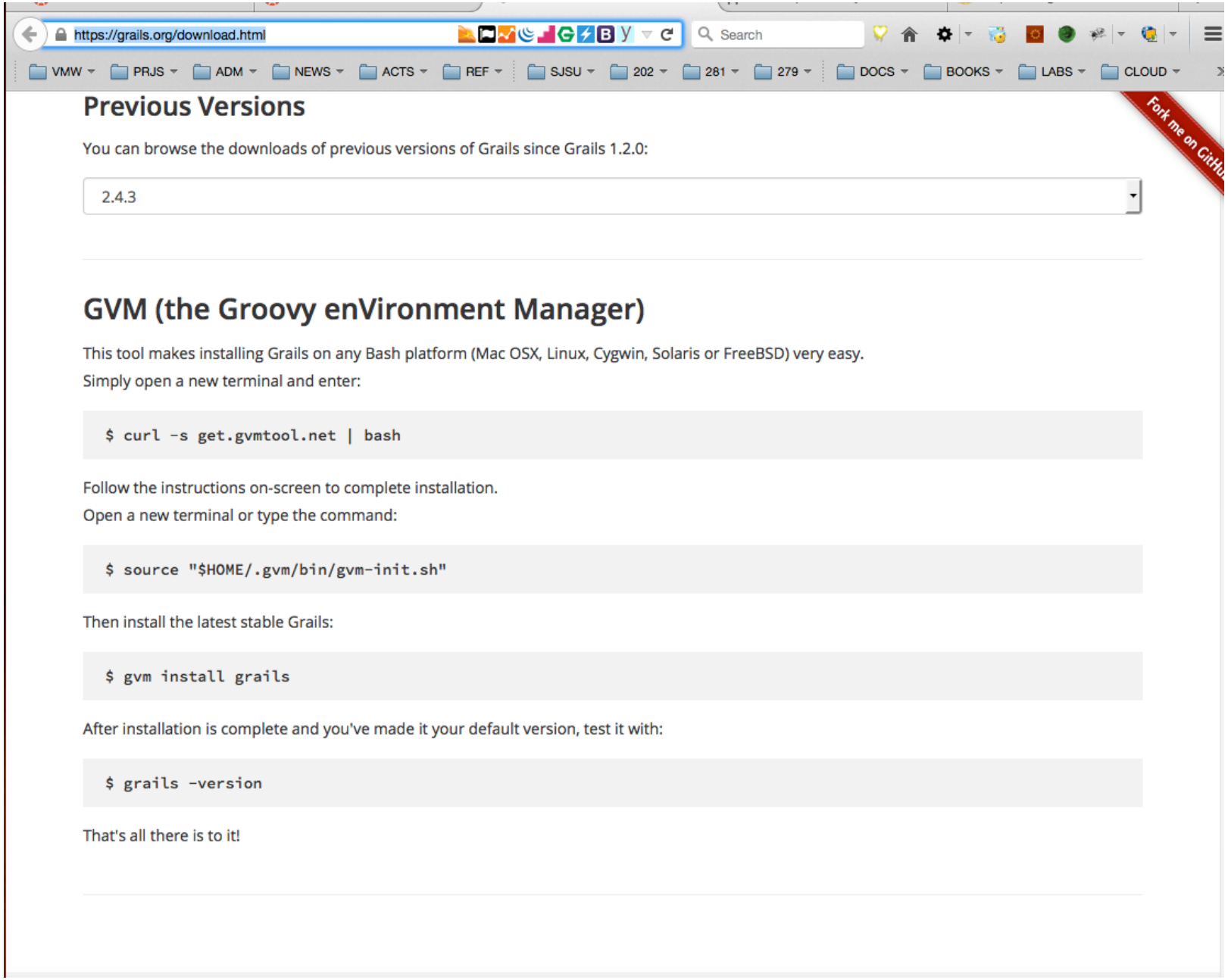
# Step #4. SSH into VM

ec2-52-11-45-67.us-west-2.compute.amazonaws.com

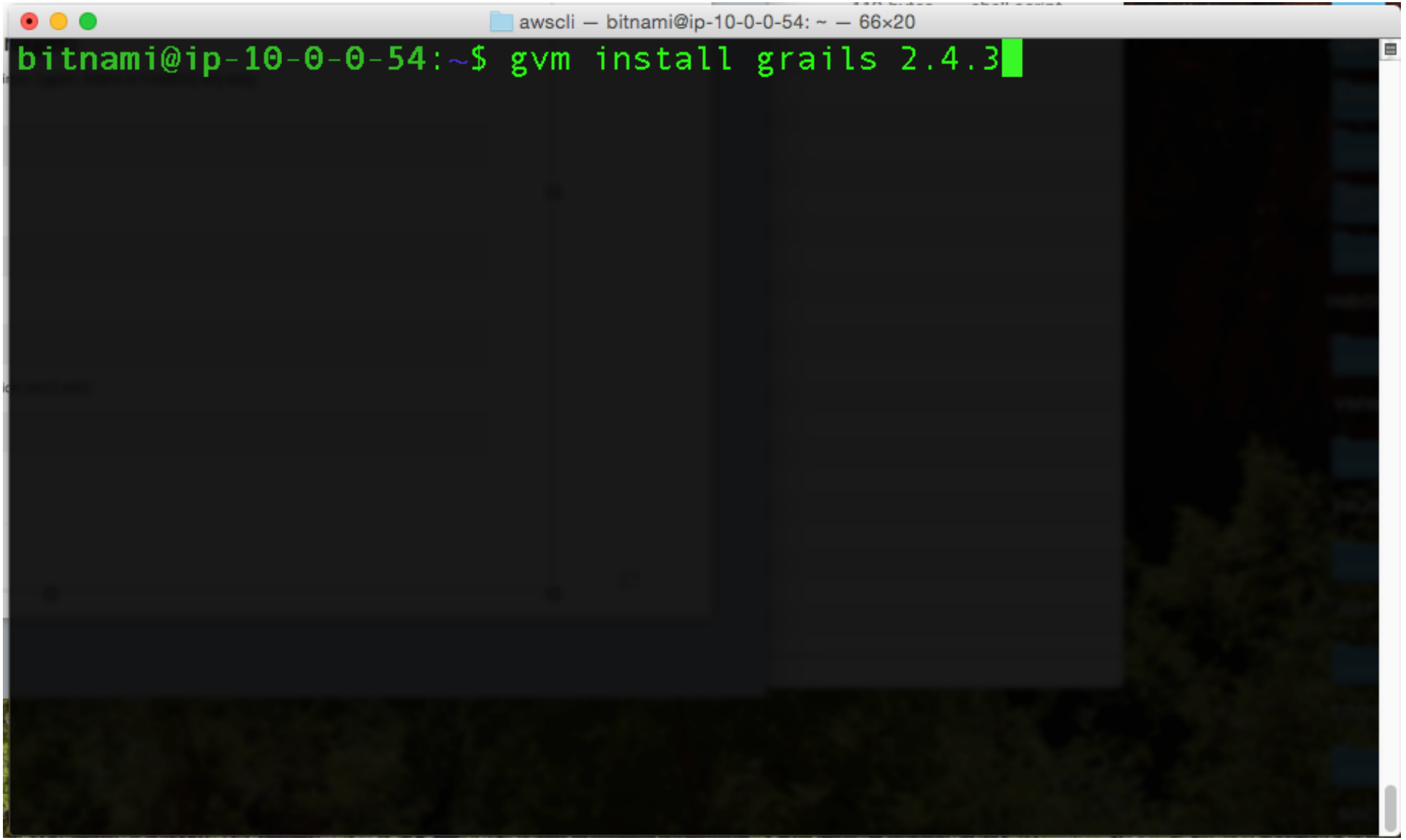


# Step #5. Setup Grails on VM

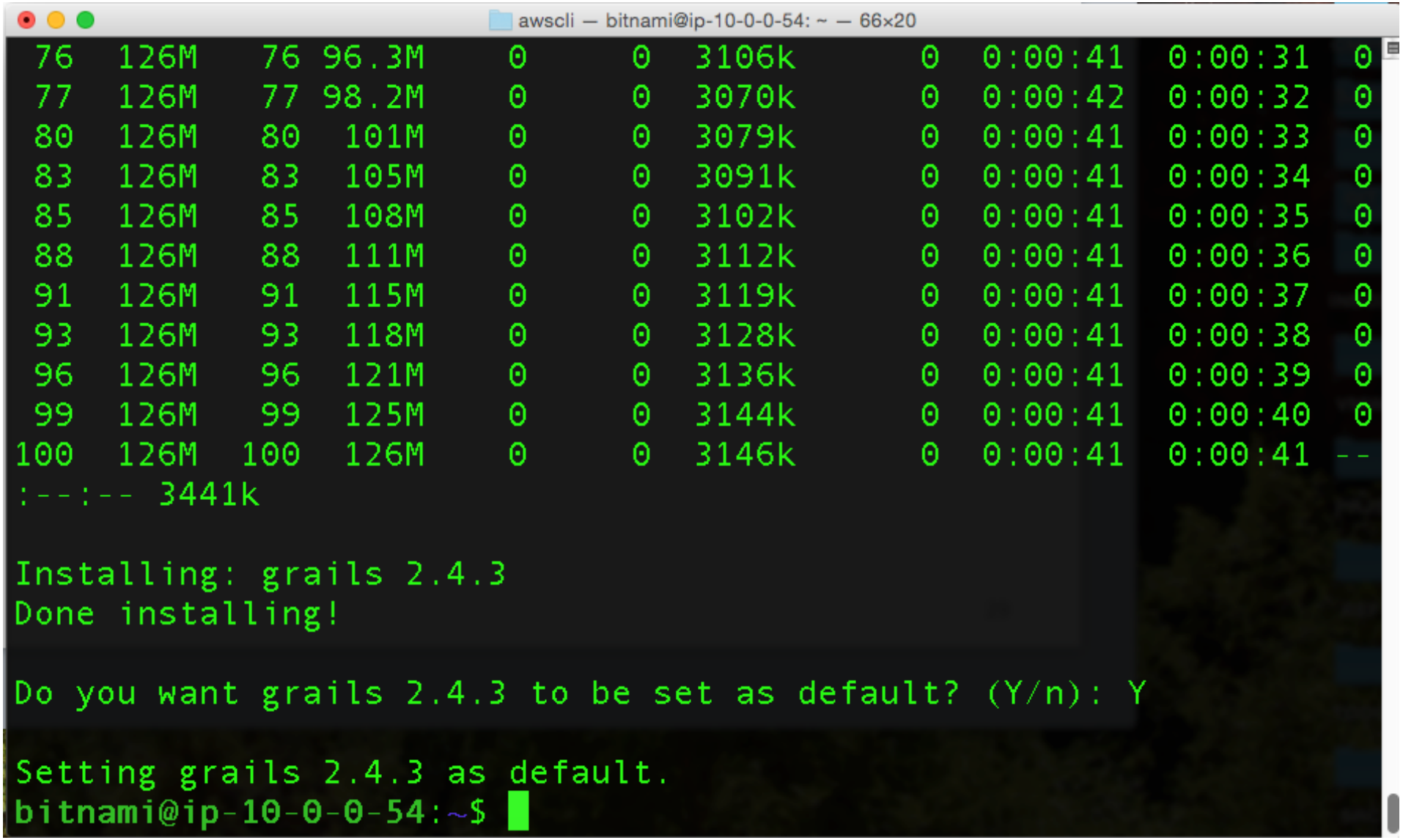
https://grails.org/download.html



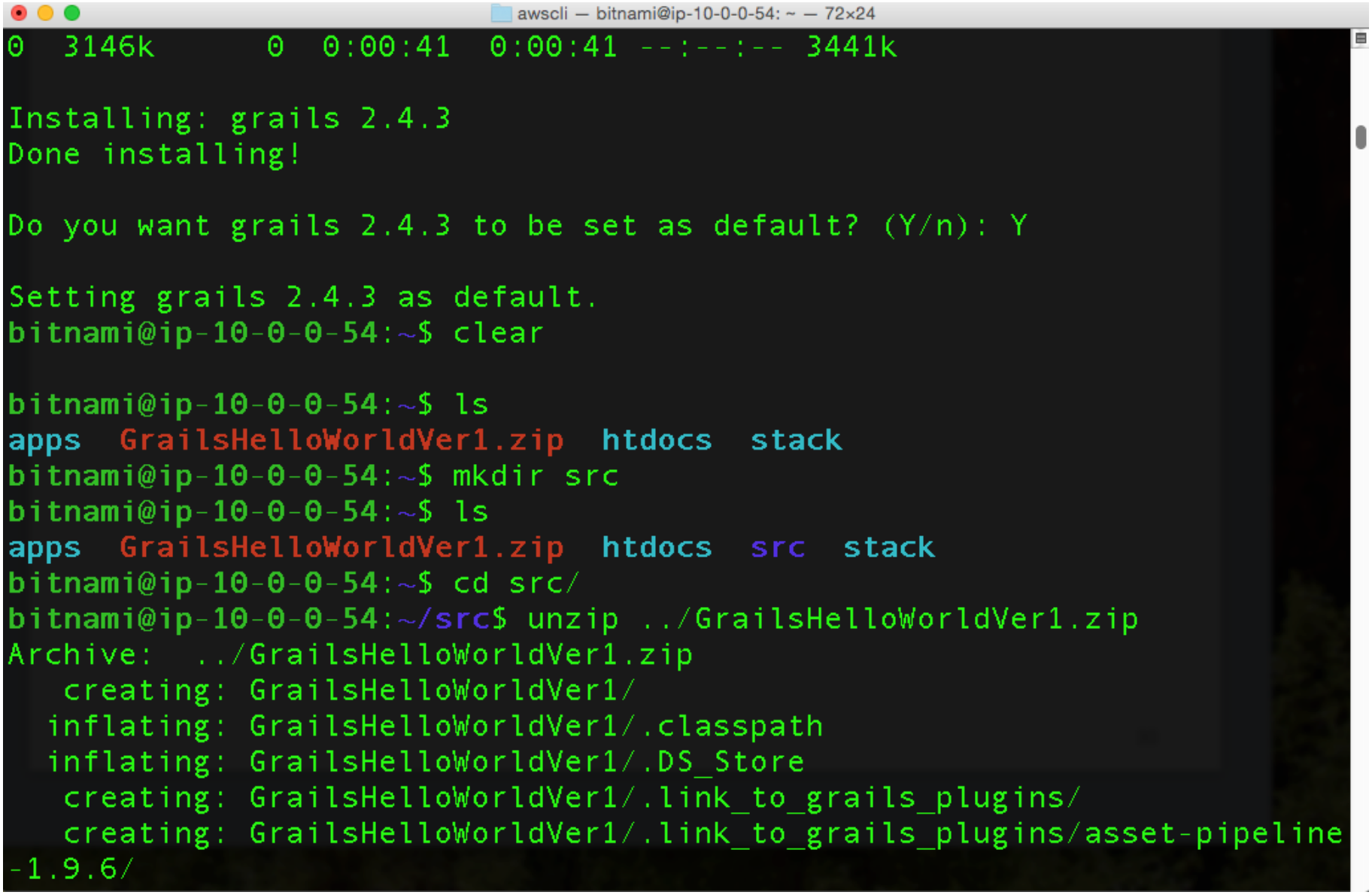
# Step #5. Setup Grails on VM



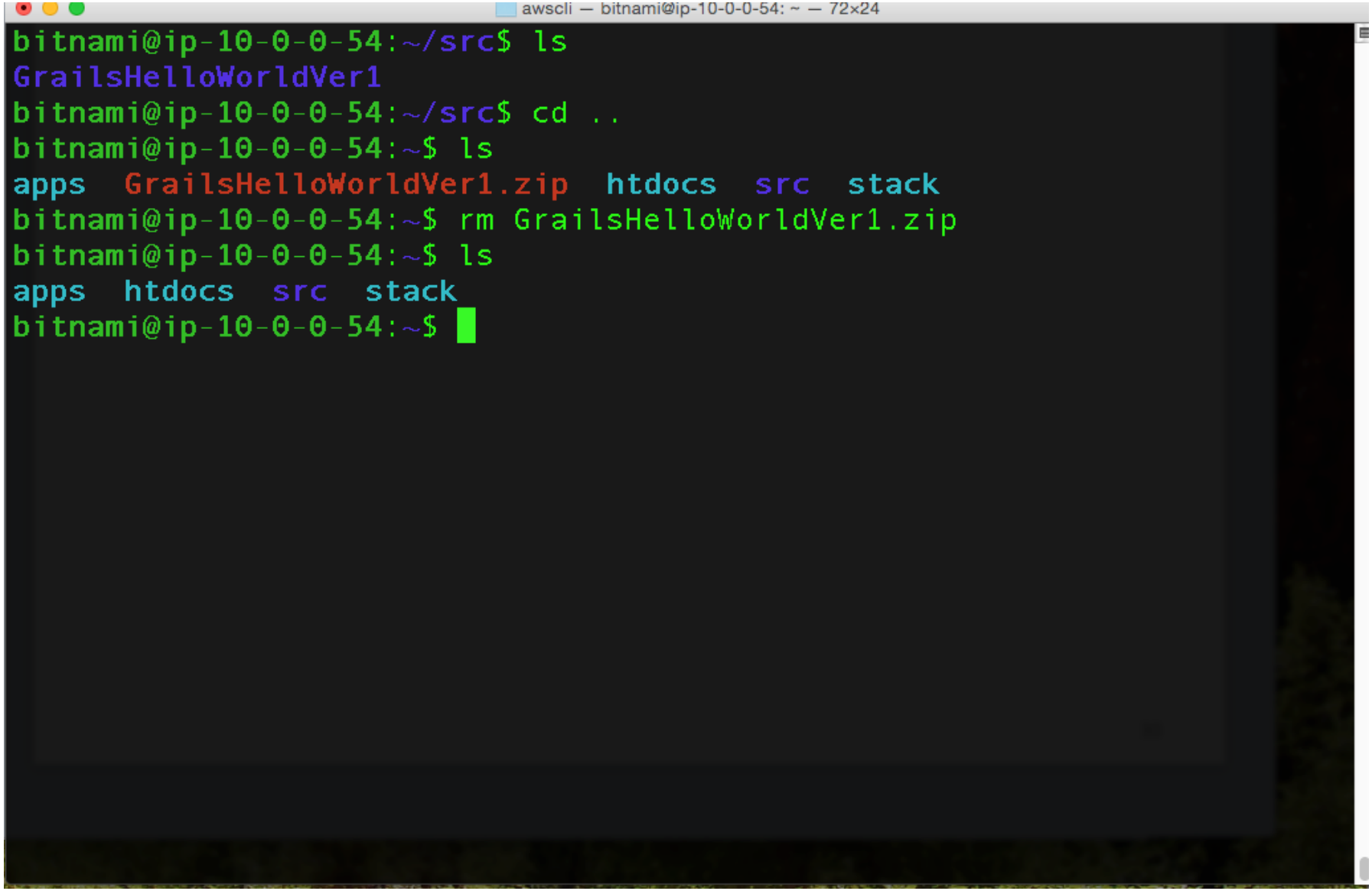
# Step #5. Setup Grails on VM



# Step 6. Build and Deploy Grails Hello World



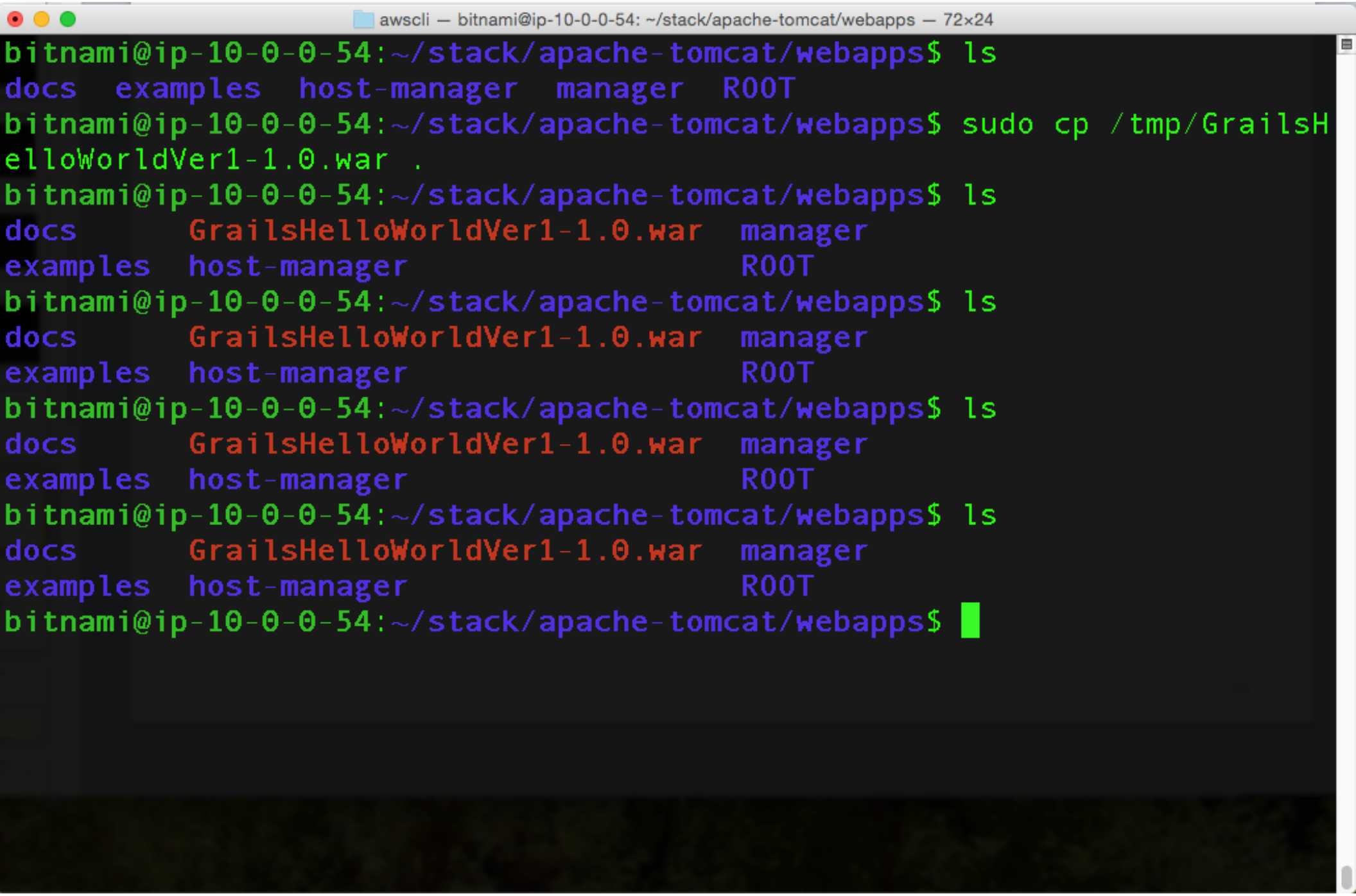
# Step 6. Build and Deploy Grails Hello World



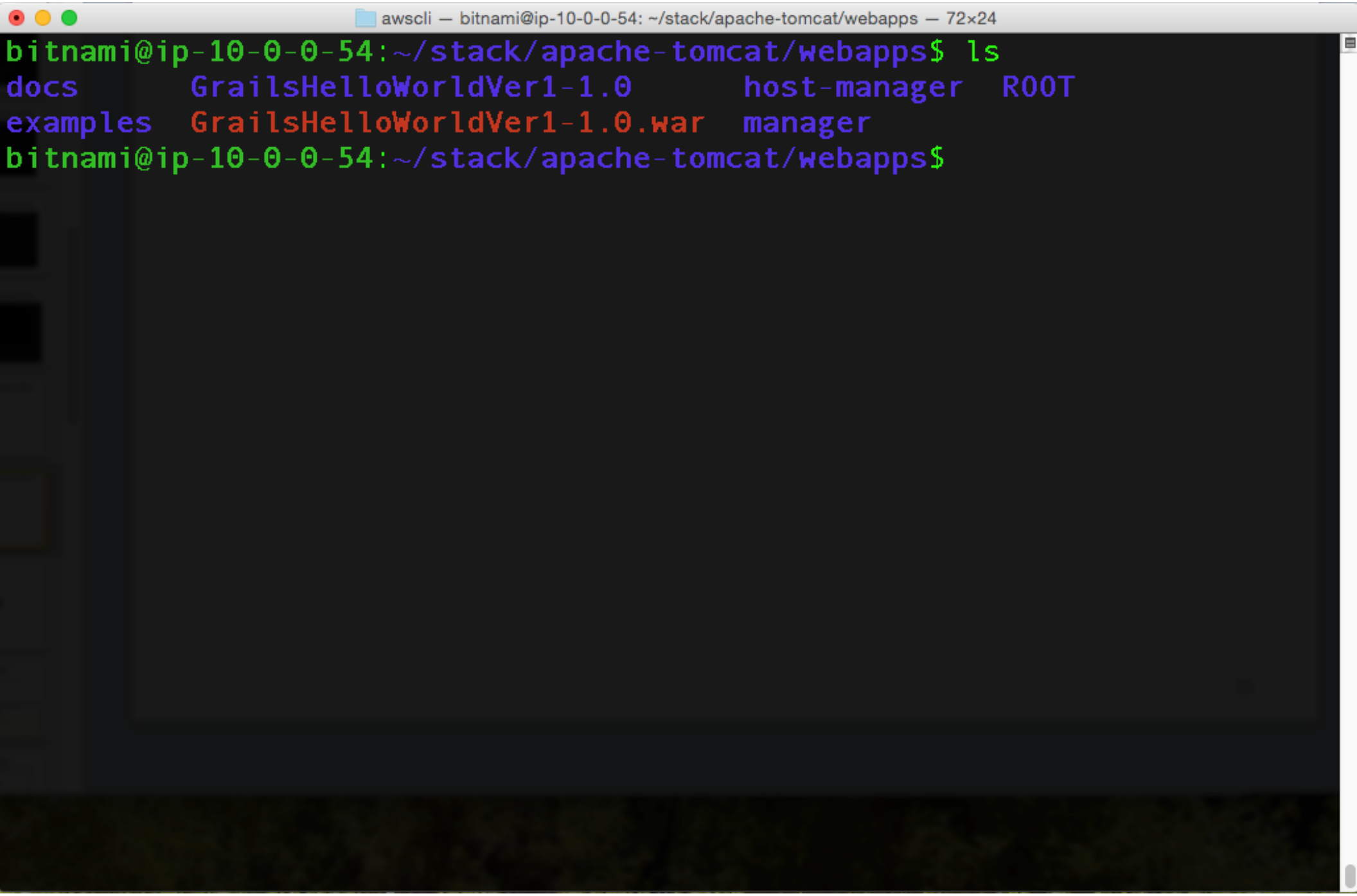
# Step 6. Build and Deploy Grails Hello World

# Step 6. Build and Deploy Grails Hello World

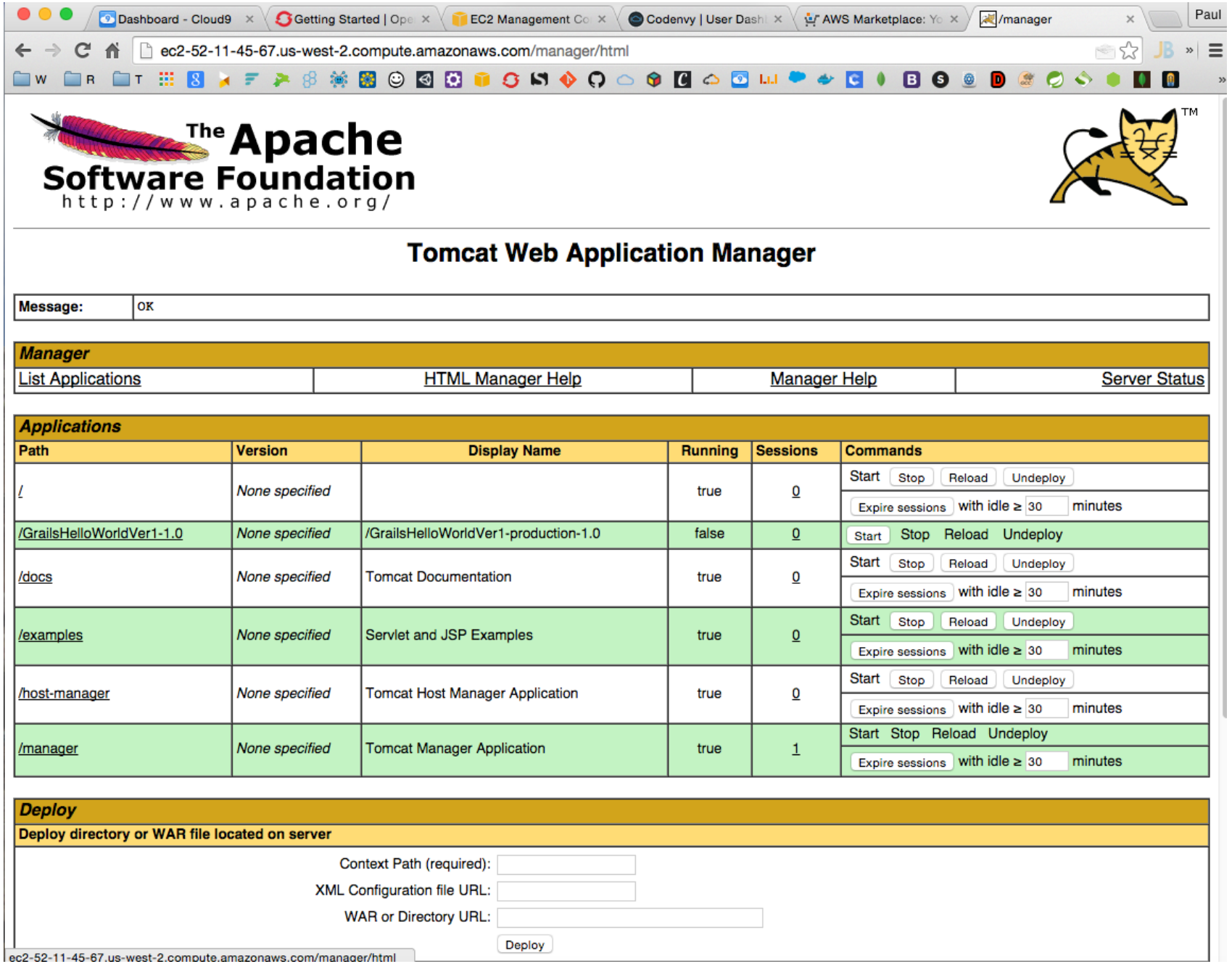
# Step 6. Build and Deploy Grails Hello World



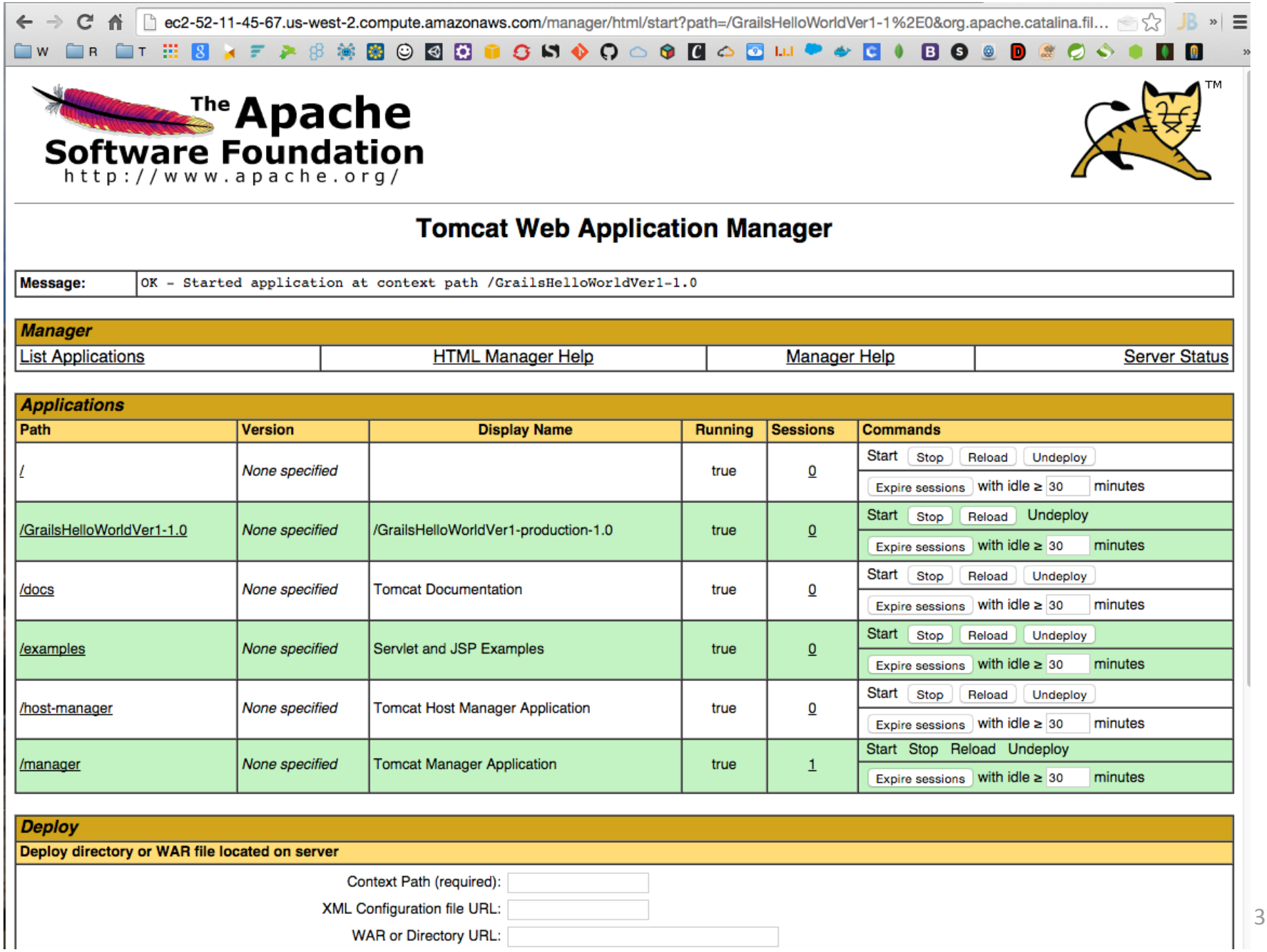
# Step 6. Build and Deploy Grails Hello World



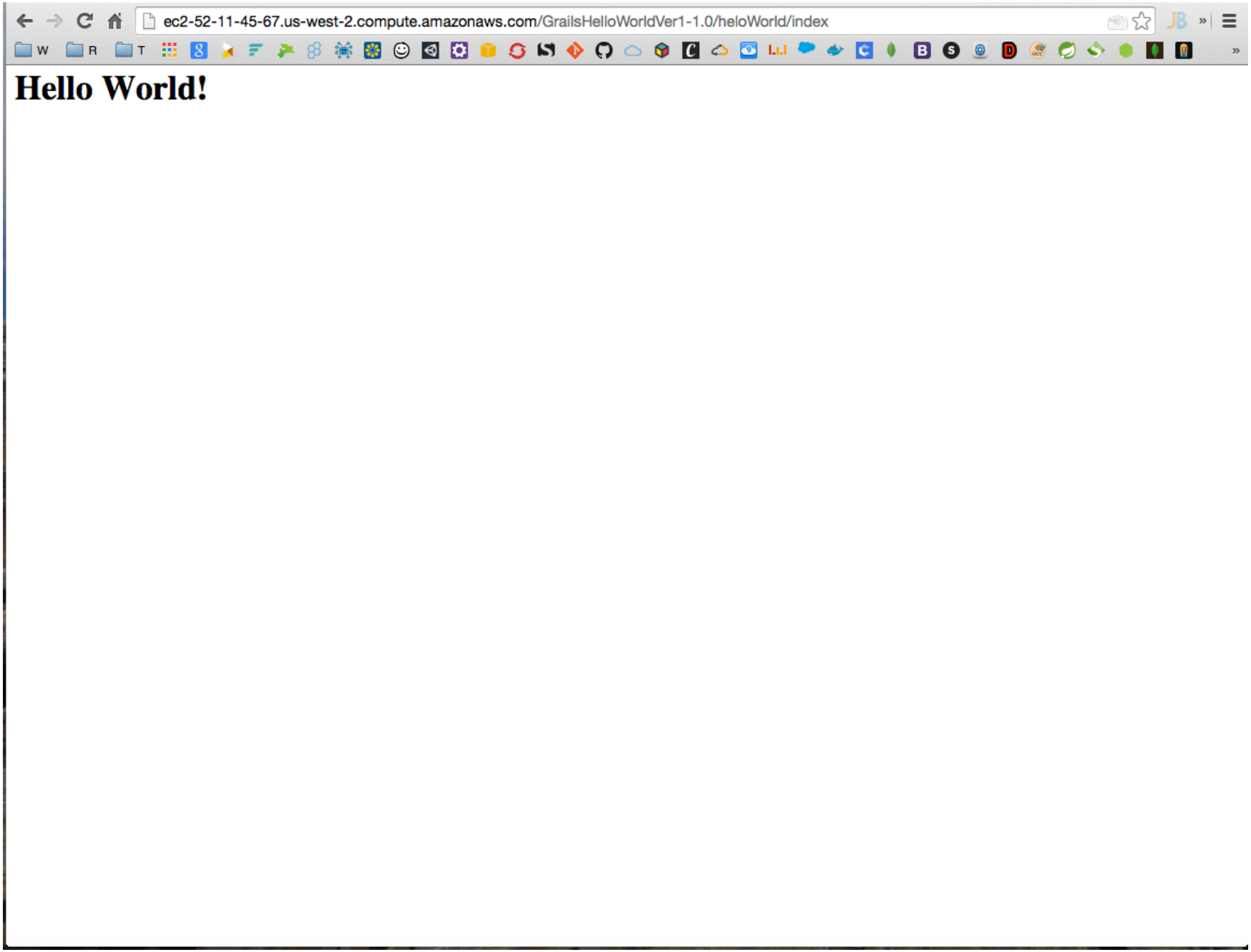
# Step 6. Build and Deploy Grails Hello World



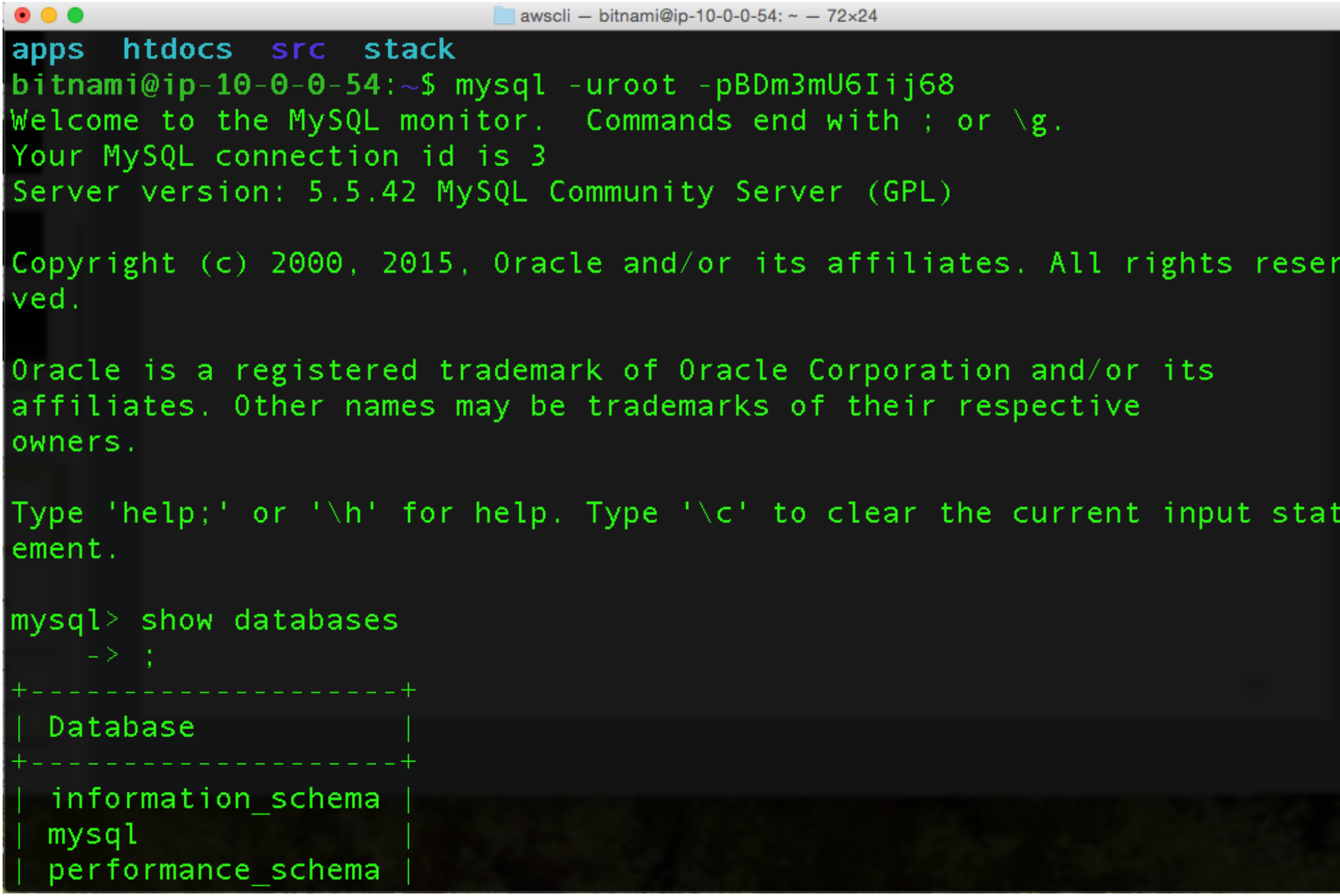
# Step 6. Build and Deploy Grails Hello World



# Step 6. Build and Deploy Grails Hello World

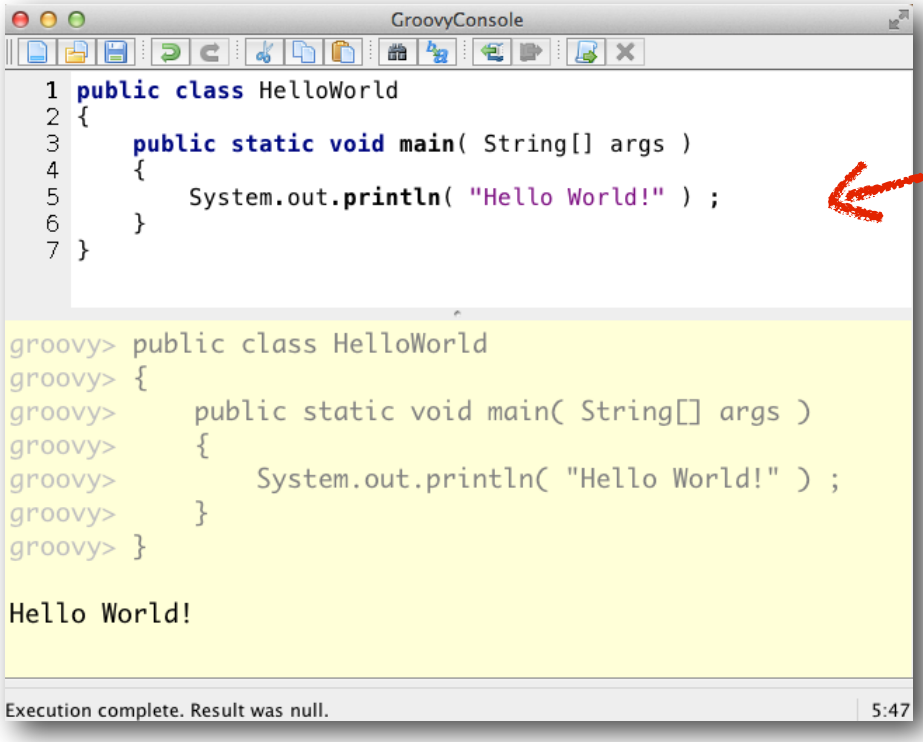


# Step 7. Access to Local MySQL (use your PWD)

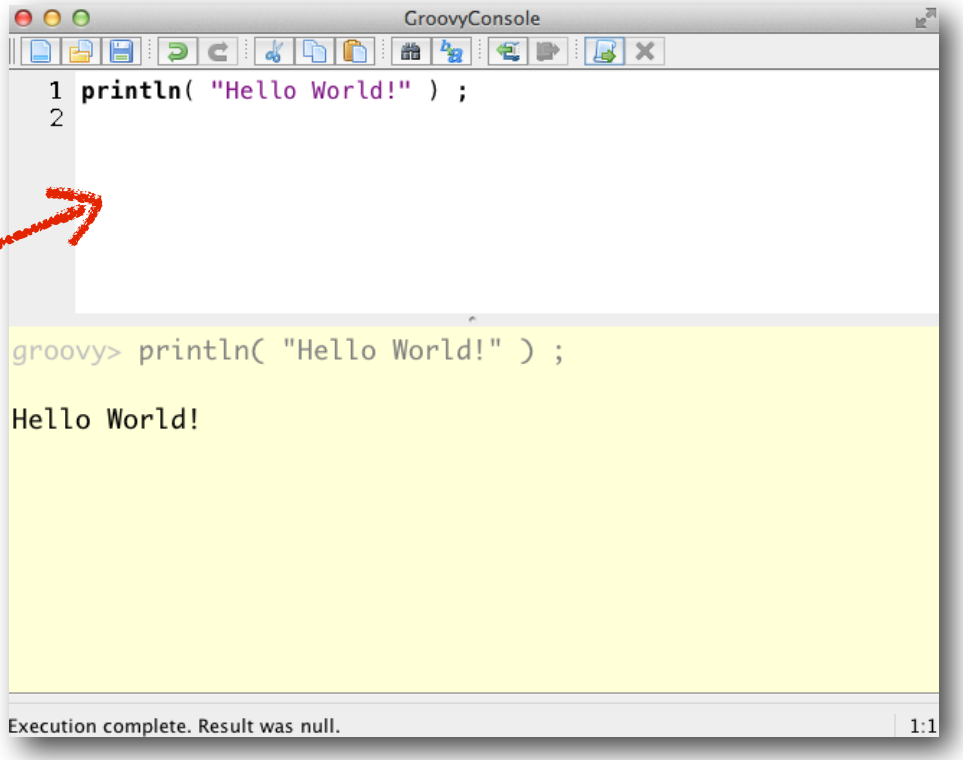


# Groovy

# Hello World

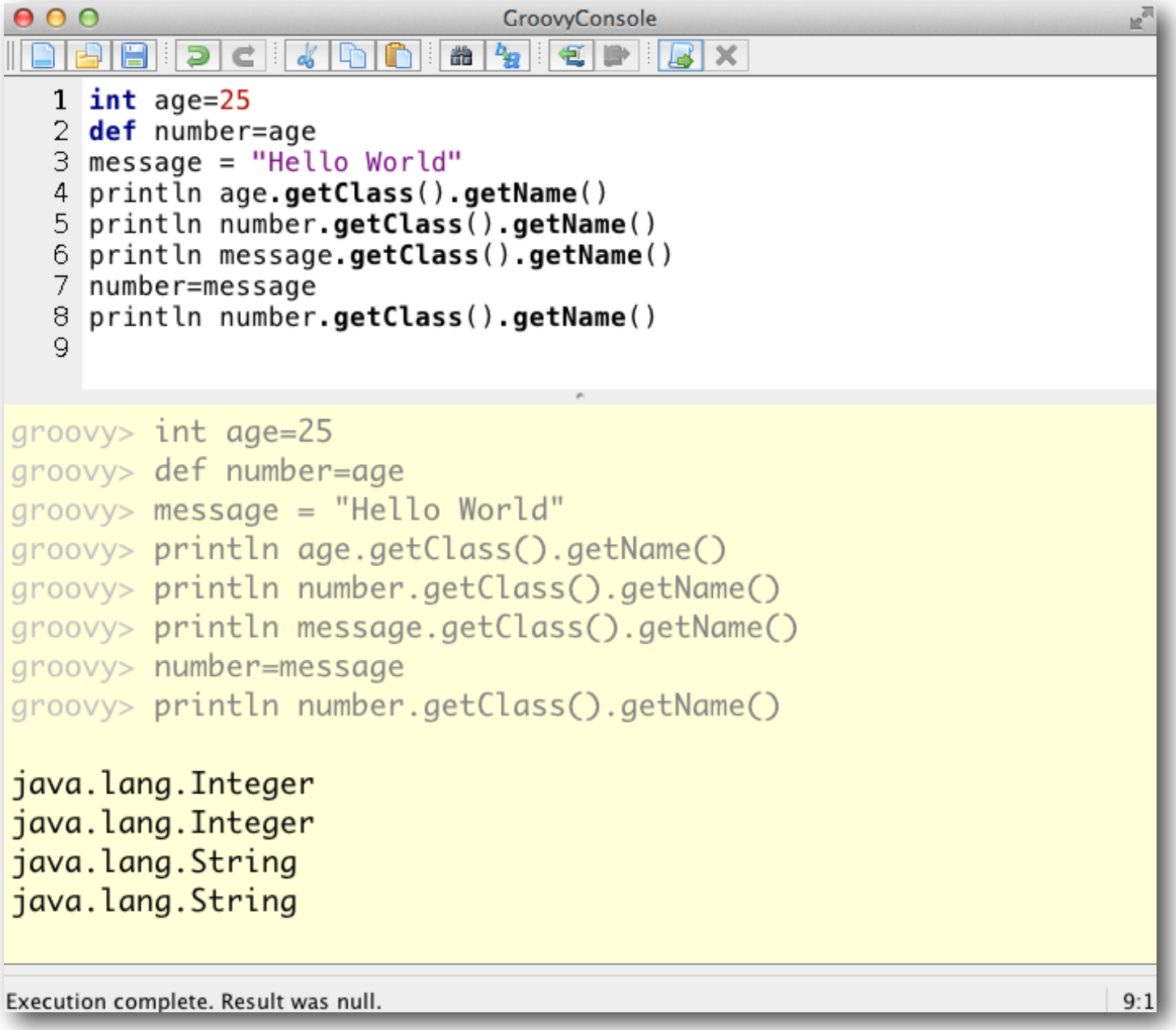



Groovy is "Java"




Yes, Groovy is "Java", but without the ceremony!

# Dynamic Types



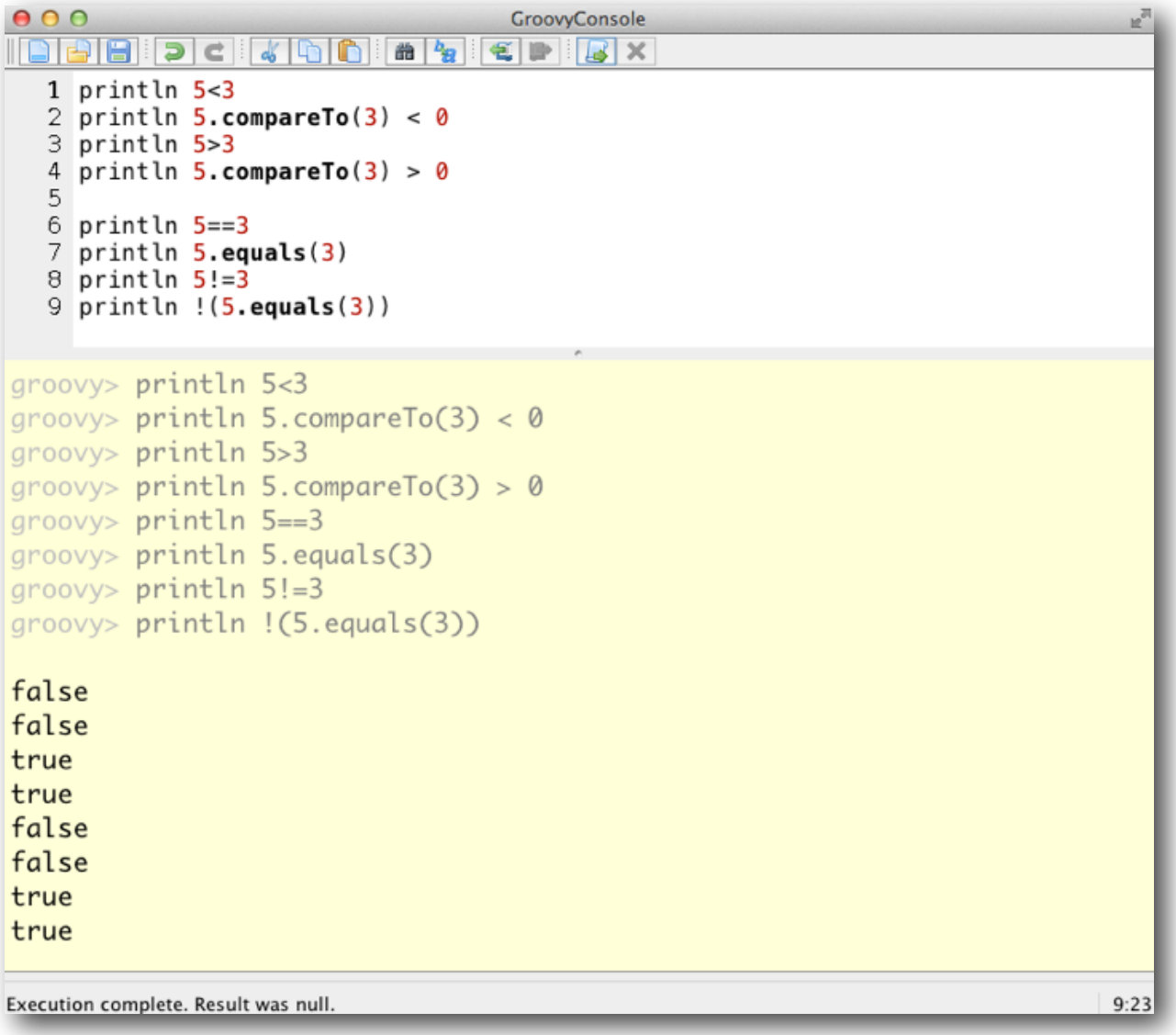
# Operators

**TABLE 2.5** Relational operators

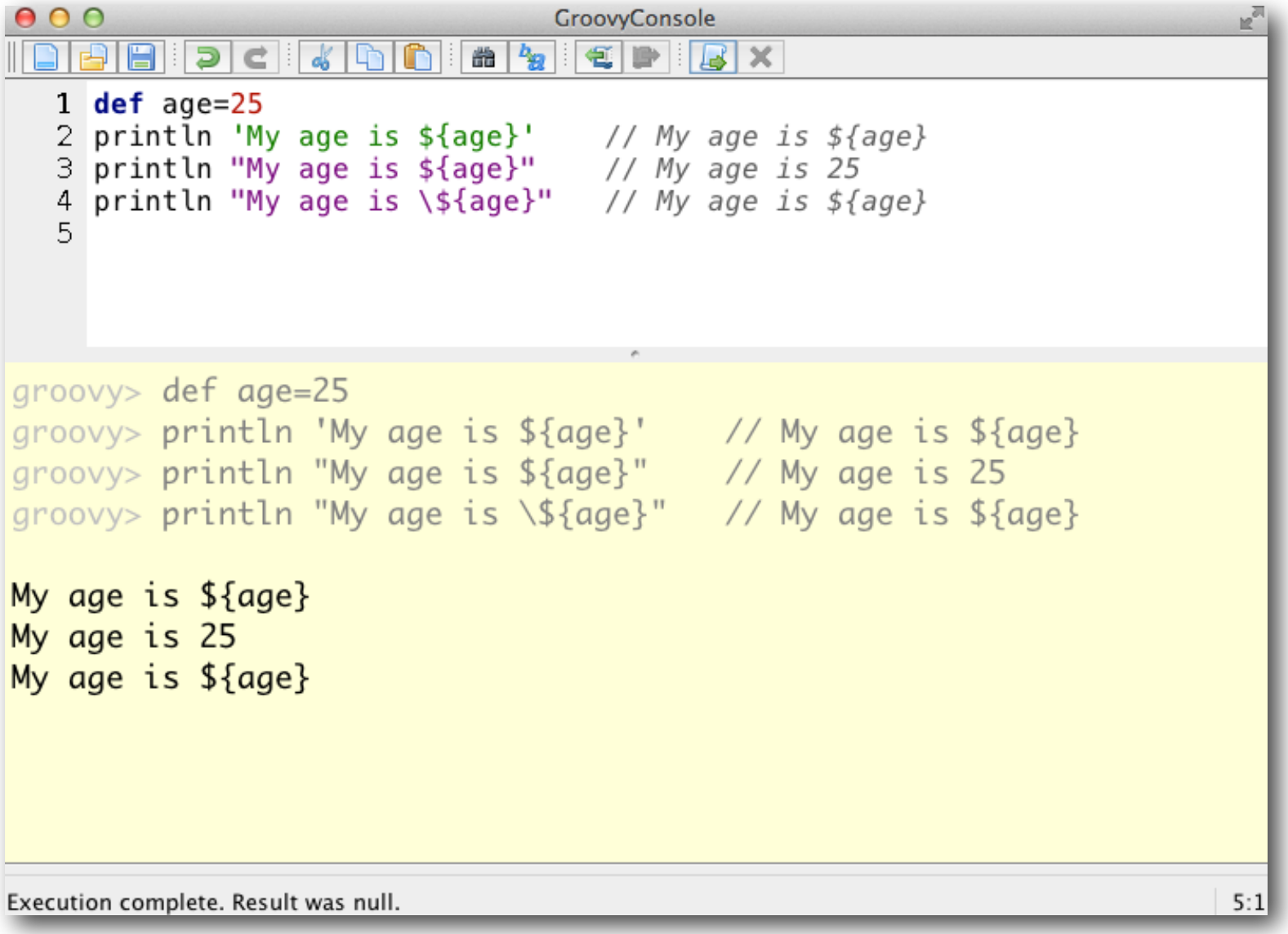
| *Expression* | *Method call* | *Result* |
|---|---|---|
| `5 < 3` | `5.compareTo(3) < 0` | `false` |
| `5 <= 3` | `5.compareTo(3) <= 0` | `false` |
| `5 > 3` | `5.compareTo(3) > 0` | `true` |
| `5 >= 3` | `5.compareTo(3) >= 0` | `true` |

**TABLE 2.6** Equality operators

| *Expression* | *Method call* | *Result* |
|---|---|---|
| `5 == 3` | `5.equals(3)` | `false` |
| `5 != 3` | `! 5.equals(3) // see Appendix C` | `true` |
| `5 <=> 3` | `5.compareTo(3)` | `+1` |



# String Literals



# Some Strings Examples

```
'Hello'.compareToIgnoreCase('hello')     // 0
'Hello'.concat('world')                  // Hello world
'Hello'.endsWith('lo')                   // true
'Hello'.equalsIgnoreCase('hello')        // true
'Hello'.indexOf('lo')                    // 3
'Hello world'.indexOf('o', 6)            // 7
'Hello'.matches('Hello')                 // true
'Hello'.matches('He')                    // false
'Hello'.replaceAll('l', 'L')             // HeLLo
'Hello world'.split('l')                 // 'He', 'o wor', 'd'
'Hello'.substring(1)                     // ello
'Hello'.substring(1, 4)                  // ell
'Hello'.toUpperCase()                    // HELLO
def message = 'Hello'
message.center(11)                       //□□□Hello□□□
message.center(3)                        // Hello
message.center(11, '#')                  // ###Hello###
message.eachMatch('.') { ch ->           // print H e l l o
  println ch }                             on separate lines
message.getAt(0)                         // H
message.getAt(0..<3)                     // Hel
message.getAt([0, 2, 4])                 // Hlo
message.leftShift('world')               // Hello world
message << 'world'                       // Hello world
message.minus('ell')                     // Ho
message-'ell'                            // Ho
message.padLeft(4)                       // Hello
message.padLeft(11)                      // □□□□□□Hello
message.padLeft(11, '#')                 // ######Hello
message.padRight(4)                      // Hello
message.padRight(11)                     // Hello□□□□□□
message.padRight(11, '#')                // Hello######
message.plus('world')                    // Hello world
message+'world'                          // Hello world
message.replaceAll('[a-z]') { ch ->      // HELLO
  ch.toUpperCase() }
message.reverse()                        // olleH
message.toList()                         // ['H', 'e', 'l', 'l', 'o
def message='Hello world'
message.tokenize()                       // ['Hello', 'world']
message.tokenize('l')                    // ['He', 'o wor', 'd']
```

```
def greeting='Hello world'
'Hello'+'world'              // Hello world       concatenate
'Hello'*3                    // HelloHelloHello   repeat
greeting-'o world'           // Hell              remove first occurrence
greeting.size()              // 11                synonymous with length
greeting.length()            // 11                synonymous with size
greeting.count('o')          // 2
greeting.contains('ell')     // true
```

In a regular expression, two special *positional characters* are used to denote the beginning and end of a line: caret (^) and dollar sign ($):

```
def rhyme='Humpty Dumpty sat on a wall'
rhyme=~'^Humpty'                          // true
rhyme=~'wall$'                            // true
```

Regular expressions can also include *quantifiers*. The plus sign (+) represents one or more times, applied to the preceding element of the expression. The asterisk (*) is used to represent zero or more occurrences. The question mark (?) denotes zero or once. The metacharacter { and } is used to match a specific number of instances of the preceding character. The following all yield `true`:

```
'aaaaab'=~'a*b'
'b'=~'a*b'
'aaacd'=~'a*c?d'
'aaad'=~'a*c?d'
'aaaaab'=~'a{5}b'
!('aab'=~'a{5}b')
```

# Collections

| *Example* | *Description* |
|---|---|
| `[11, 12, 13, 14]` | A list of integer values |
| `['Ken', 'John', 'Andrew']` | A list of `Strings` |
| `[1, 2, [3, 4], 5]` | A nested list |
| `['Ken', 21, 1.69]` | A heterogeneous list of object references |
| `[ ]` | An empty list |

```
def numbers = [11, 12, 13, 14]     // list with four items
numbers [0]                        // 11
numbers [3]                        // 14

numbers [-1]   // 14
numbers [-2]   // 13

[11, 12, 13, 14][2]   // 13

numbers [0..2]    // [11, 12, 13]
numbers [1..<3]   // [12, 13]

numbers [1] = 22          // [11, 22, 13, 14]
numbers [1] = [33, 44]    // [11, [33, 44], 13, 14]

numbers << 15     // [11, [33, 44], 13, 14, 15]

numbers = [11, 12, 13, 14]     // list with four items
numbers + [15, 16]             // [11, 12, 13, 14, 15, 16]

numbers = [11, 12, 13, 14]     // list with four items
numbers - [13]                 // [11, 12, 14]
```

**TABLE 4.2** List methods

| *Name* | *Signature/description* |
|---|---|
| `add` | `boolean add(Object value)`<br>Append the new `value` to the end of this `List`. |
| `add` | `void add(int index, Object value)`<br>Inserts a new `value` into this `List` at the given `index` position. |
| `addAll` | `boolean addAll(Collection values)`<br>Append the new `values` on to the end of this `List`. |
| `contains` | `boolean contains(Object value)`<br>Returns `true` if this `List` contains the specified `value`. |
| `flatten` * | `List flatten()`<br>Flattens this `List` and returns a new `List`. |
| `get` | `Object get(int index)`<br>Returns the element at the specified position in this `List`. |

```
[11, 12, 13, 14].add(15)                  // [11, 12, 13, 14, 15]
[11, 12, 13, 14].add(2, 15)               // [11, 12, 15, 13, 14]
[11, 12, 13, 14].add([15, 16])            // [11, 12, 13, 14, 15, 16]
[11, 12, 13, 14].get(1)                   // 12
[11, 12, 13, 14].isEmpty()                // false
[14, 13, 12, 11].size()                   // 4
[11, 12, [13, 14]].flatten()              // [11, 12, 13, 14]
[11, 12, 13, 14].getAt(1)                 // 12
[11, 12, 13, 14].getAt(1..2)              // [12, 13]
[11, 12, 13, 14].getAt([2, 3])            // [13, 14]
[11, 12, 13, 14].intersect([13, 14, 15])  // [13, 14]
[11, 12, 13, 14].pop()                    // 14
[11, 12, 13, 14].reverse()                // [14, 13, 12, 11]
[14, 13, 12, 11].sort()                   // [11, 12, 13, 14]
```

# Maps

| *Example* | *Description* |
|---|---|
| `['Ken' : 'Barclay', 'John' : 'Savage']` | Forename/surname collection |
| `[4 : [2], 6 : [3, 2], 12 : [6, 4, 3, 2]]` | Integer keys and their list of divisors |
| `[ : ]` | Empty map |



```
def names = ['Ken' : 'Barclay', 'John' : 'Savage']
def divisors = [4 : [2], 6 : [3, 2], 12 : [6, 4, 3, 2]]
names['Ken']                                    // 'Barclay'
names.Ken                                       // 'Barclay'
names['Jessie']                                 // null
divisors[6]                                     // [3, 2]
```

```
divisors[6] = [6, 3, 2, 1]          // [4 : [2], 6 : [6, 3, 2, 1],
                                    // 12 : [6, 4, 3, 2]]
```

```
def careful = [ 1 : 'Ken', '1' : 'Barclay']
careful[1]                                  // Ken
careful['1']                                // Barclay
```

**TABLE 4.4** Map methods

| *Name* | *Signature/description* |
|---|---|
| `containsKey` | `boolean containsKey(Object key)`<br>Does this `Map` contain this `key`? |
| `get` | `Object get(Object key)`<br>Look up the `key` in this `Map` and return the corresponding value. If there is no entry in this `Map` for the key, then return `null`. |
| `get` * | `Object get(Object key, Object defaultValue)`<br>Look up the `key` in this `Map` and return the corresponding value. If there is no entry in this `Map` for the key, then return the `defaultValue`. |
| `getAt` * | `Object getAt(Object key)`<br>Support method for the subscript operator. |
| `keySet` | `Set keySet()`<br>Obtain a `Set` of the keys in this `Map`. |
| `put` | `Object put(Object key, Object value)`<br>Associates the specified `value` with the specified `key` in this `Map`. If this `Map` previously contained a mapping for this `key`, the old value is replaced by the specified `value`. |
| `putAt` * | `Object putAt(Object key, Object value)`<br>Support method to allow `Maps` to operate with subscript assignment. |
| `size` | `int size()`<br>Returns the number of key-value mappings in this `Map`. |
| `values` | `Collection values()`<br>Returns a collection view of the values contained in this `Map`. |

```
def mp = ['Ken' : 2745, 'John' : 2746, 'Sally' : 2742]
mp.put('Bob', 2713)         // [Bob:2713, Ken:2745, Sally:2742, John:2746]
mp.containsKey('Ken')       // true
mp.get('David', 9999)       // 9999
mp.get('Sally')             // 2742
mp.get('Billy')             // null
mp.keySet()                 // [David, Bob, Ken, Sally, John]
mp.size()                   // 4
mp['Ken']                   // 2745
```

# Methods

```
def methodName() {
   // Method code goes here
}
```

```
def methodName(para1, para2, para3) {
  // Method code goes here
}
```

```
def someMethod(para1, para2=0, para3=0) {
  // Method code goes here
}
```

```
def greetings() {
    print 'Hello'
    print ' and '
    println 'welcome'
}

greetings()
```

```
def greetings(name) {
  println "Hello and welcome, ${name}"
}

greetings('John')
```

```
def greetings(salutation, name='Ken') {
  println "${salutation} ${name}"
}
greetings('Hello', 'John')        // Hello John
greetings('Welcome')              // Welcome Ken
```

```
Hello and welcome
```

```
Hello and welcome, John
```

```
Hello John
Welcome Ken
```

# Closures - Invocation & Parameters

## 9.1 CLOSURES

The syntax for defining a closure is:

```
{comma-separated-formal-parameter-list -> statement-list}
```

If no formal parameters are required, then the parameter `List` and the `->` separator are omitted. Here is a simple example of a closure with no parameters.

```
def clos={println 'Hello world'}
clos.call()
```

```
Hello world
```

```
def clos={param -> println "Hello ${param}"}

clos.call('world')        // actual argument is 'world'
clos.call('again')        // actual argument is 'again'
clos('shortcut')          // abbreviated form
```

```
Hello world
Hello again
Hello shortcut
```

Observe the third invocation in which the `call` has been omitted.

The next illustration repeats the previous example and produces the same result, but shows that an implicit single parameter referred to as `it` can be used.

```
def clos={println "Hello ${it}"}

clos.call('world')
clos.call('again')

clos('shortcut')
```

Default Single Parameter
"it"

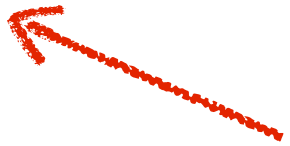
# Closures - Enclosing Scope


```
def greeting = 'Hello'
def clos = {param -> println "${greeting} ${param}"}
clos.call('world')

// Now show that changes to this variable change the closure.
greeting = 'Welcome'
clos.call('world')

def demo(clo) {
    def greeting = 'Bonjour'                // does not affect closure
    clo.call('Ken')
}

demo(clos)
```




The resulting output is:

```
Hello world
Welcome world
Welcome Ken
```

# Groovy Quick Start

## Overview

Groovy classes compile down to Java bytecode and so there's a 1-1 mapping between a Groovy class and a Java class.

Indeed each Groovy class can be used inside normal Java code - since it is a Java class too.

Probably the easiest way to get groovy is to try working with collections. In Groovy List (java.util.List) and Map (java.util.Map) are both first class objects in the syntax. So to create a List of objects you can do the following...

```
def list = [1, 2, 'hello', new java.util.Date()]
assert list.size() == 4
assert list.get(2) == 'hello'
assert list[2] == 'hello'
```

Notice that everything is an object (or that auto-boxing takes place when working with numbers). To create maps...

```
def map = ['name':'James', 'location':'London']
assert map.size() == 2
assert map.get('name') == 'James'
assert map['name'] == 'James'
```

Iterating over collections is easy...

```
def list = [1, 2, 3]
for (i in list) { println i }
```

*http://groovy.codehaus.org/Quick+Start*

# Groovy Quick Start

## Working with closures

Closures are similar to Java's inner classes, except they are a single method which is invokable, with arbitrary parameters. A closure can have as many parameters as you wish...

```
def closure = { param -> println("hello ${param}") }
closure.call("world!")

closure = { greeting, name -> println(greeting + name) }
closure.call("hello ", "world!")
```

If no parameter(s) is(are) specified before -> symbol then a default named parameter, called 'it' can be used. e.g.

```
def closure = { println "hello " + it }
closure.call("world!")
```

Using closures allows us to process collections (arrays, maps, strings, files, SQL connections and so forth) in a clean way. e.g

```
[1, 2, 3].each ({ item -> print "${item}-" })
["k1":"v1", "k2":"v2"].each {key, value -> println key + "=" + value}
```

*http://groovy.codehaus.org/Quick+Start*

# Closures & Collections

Several List, Map, and String methods accept a closure as an argument (see also Appendix H). It is this combination of closures and collections that provides Groovy with some elegant solutions to common programming problems. For example, the each method with the signature:

```
void each(Closure closure)
```

is used to iterate through a List, Map, or String and apply the closure on every element. Example 08 presents a number of simple examples of the each method and a closure.

```
[1, 2, 3, 4].each {println it}

['Ken' : 21, 'John' : 22, 'Sally' : 25].each {println it}
['Ken' : 21, 'John' : 22, 'Sally' : 25].each {println "${it.key} maps to: ${it.value}"}

'Ken'.each {println it}
```

```
1
2
3
4
Sally=25
John=22
Ken=21
Sally maps to: 25
John maps to: 22
Ken maps to: 21
K
e
n
```

# Groovy Quick Start

**Note:** If a given closure is the last parameter of a method, its definition can reside outside of the parentheses. Thus the following code is valid:

```
def fun(int i, Closure c) {
  c.call(i)
}

// put Closure out of ()
[1, 2, 3].each() { item -> print "${item}-" } // 1-2-3-
fun(123) { i -> println i } // 123

// omit ()
[1, 2, 3].each ({ item -> print "${item}-" }) // 1-2-3-

// omit enclosing ()
[1, 2, 3].each { item -> print "${item}-" } // 1-2-3-

// normal
[1, 2, 3].each({ item -> print "${item}-" }) // 1-2-3-

// using the fun function to do the same thing
[1,2,3].each {fun(it,{item -> print "${item}-"})} // 1-2-3-

def closure = { i -> println i}

//[1, 2, 3].each() closure // error. closure has been previously defined
```

http://groovy.codehaus.org/Quick+Start

# Groovy Quick Start

## each

iterate via a closure

```
[1, 2, 3].each { item -> print "${item}-" }
```

## collect

collect the return value of calling a closure on each item in a collection

```
def value = [1, 2, 3].collect { it * 2 }
assert value == [2, 4, 6]
```

## find

finds first item matching closure predicate

```
def value = [1, 2, 3].find { it > 1 }
assert value == 2
```

## findAll

finds all items matching closure predicate

```
def value = [1, 2, 3].findAll { it > 1 }
assert value == [2, 3]
```

http://groovy.codehaus.org/Quick+Start

# Groovy Quick Start

## inject

allows you to pass a value into the first iteration and then pass the result of that iteration into the next iteration and so on. This is ideal for counting and other forms of processing

```
def value = [1, 2, 3].inject('counting: ') { str, item -> str + item }
assert value == "counting: 123"

value = [1, 2, 3].inject(0) { count, item -> count + item }
assert value == 6
```

In addition there's 2 new methods for doing boolean logic on some collection...

## every

returns true if all items match the closure predicate

```
def value = [1, 2, 3].every { it < 5 }
assert value

value = [1, 2, 3].every { item -> item < 3 }
assert ! value
```

## any

returns true if any item match the closure predicate

```
def value = [1, 2, 3].any { it > 2 }
assert value

value = [1, 2, 3].any { item -> item > 3 }
assert value == false
```

*http://groovy.codehaus.org/Quick+Start*

# Groovy Quick Start

### max / min

returns the max/min values of the collection - for Comparable objects

```
value = [9, 4, 2, 10, 5].max()
assert value == 10
value = [9, 4, 2, 10, 5].min()
assert value == 2
value = ['x', 'y', 'a', 'z'].min()
assert value == 'a'
```

### join

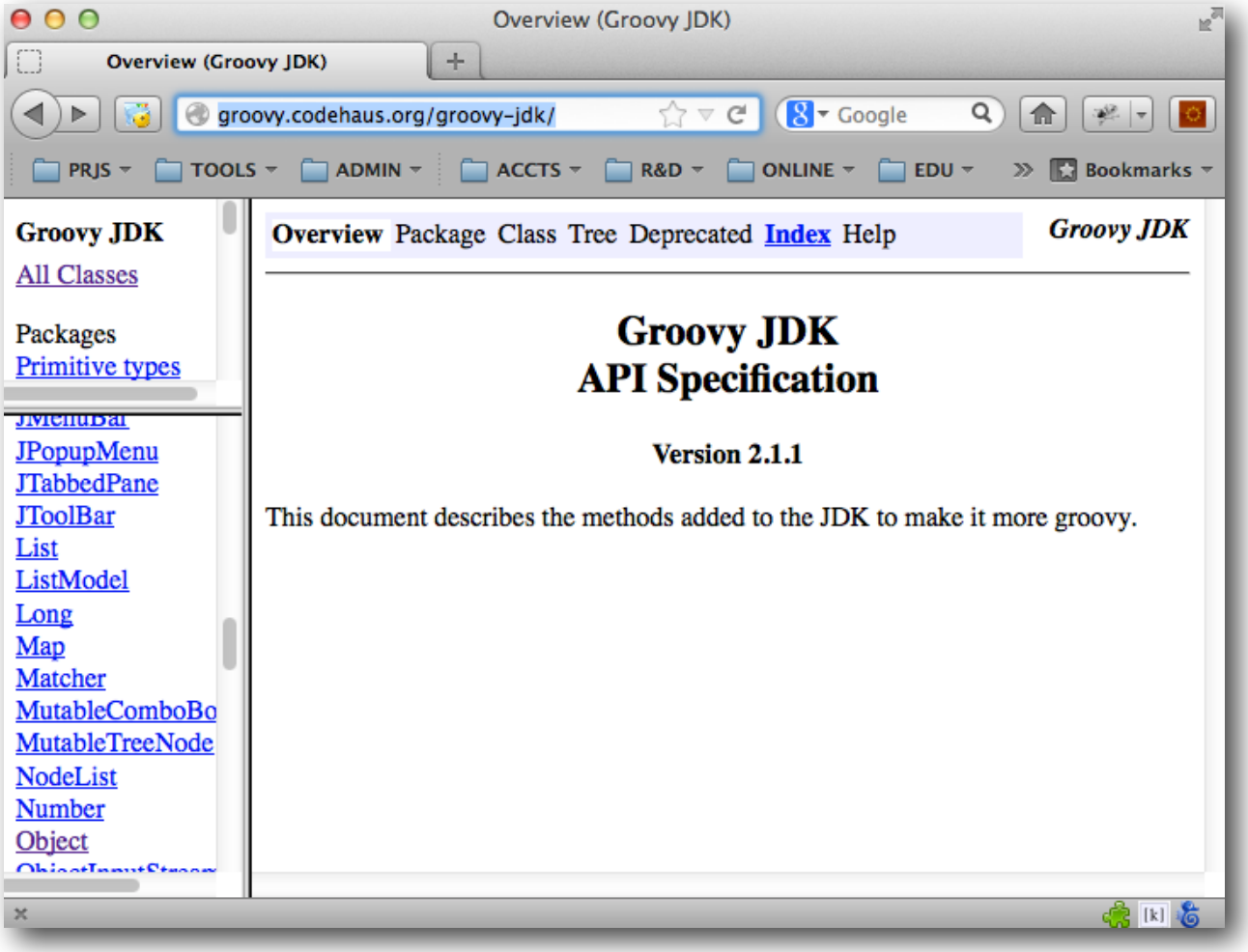
concatenates the values of the collection together with a string value

```
def value = [1, 2, 3].join('-')
assert value == '1-2-3'
```

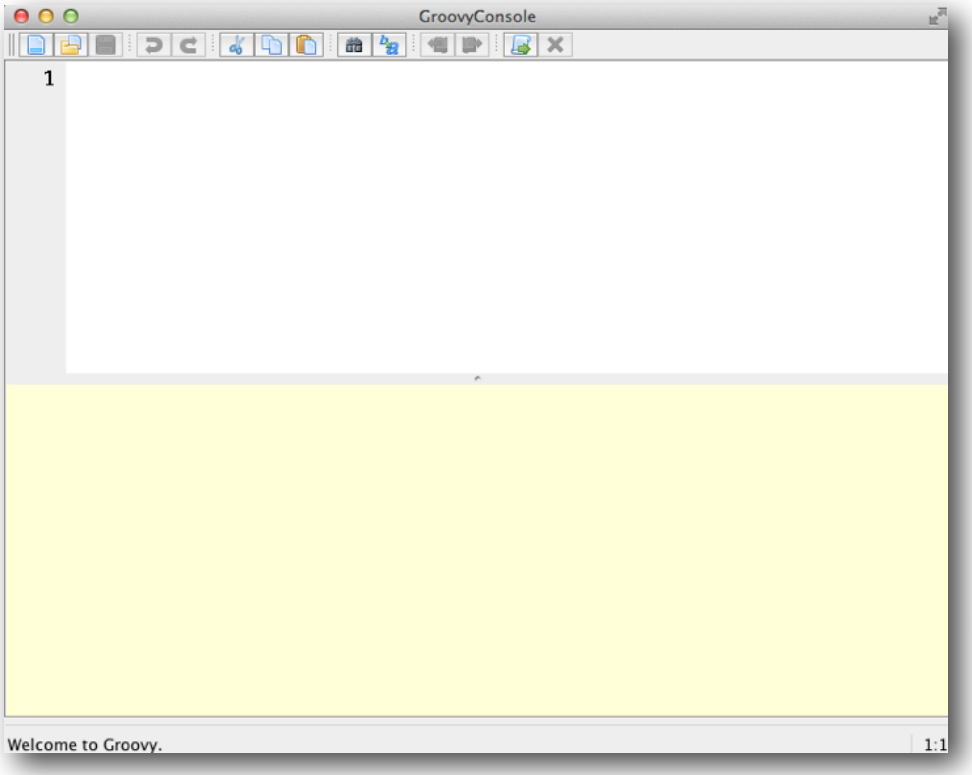
http://groovy.codehaus.org/Quick+Start

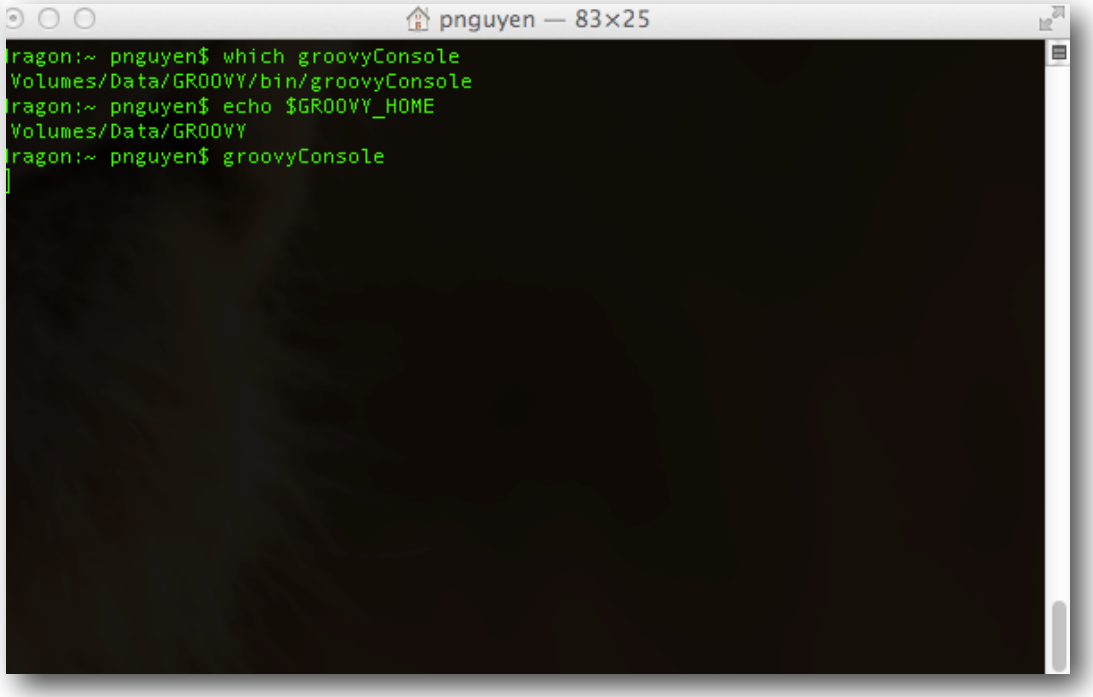
# GDK

http://groovy.codehaus.org/groovy-jdk/



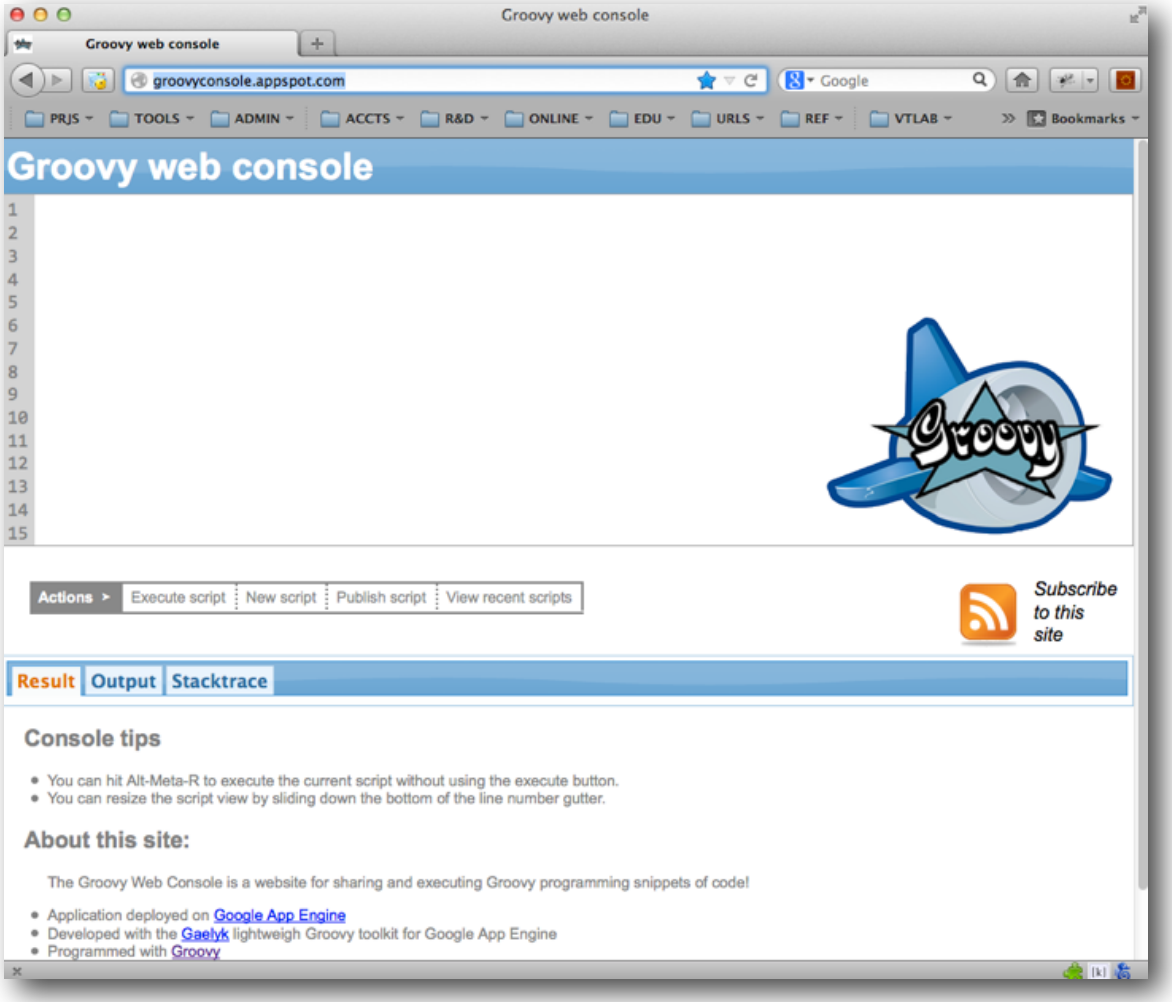
# Groovy Console



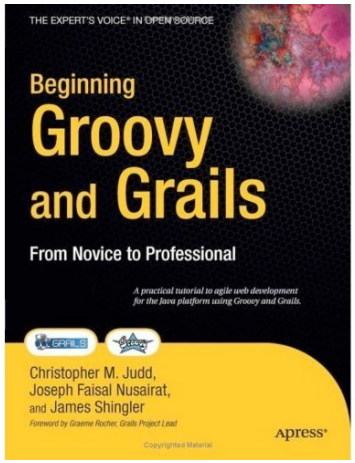



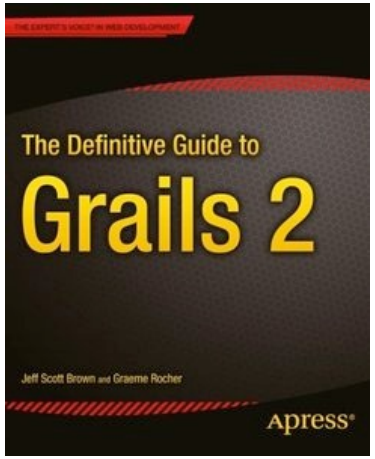
# Groovy Web Console

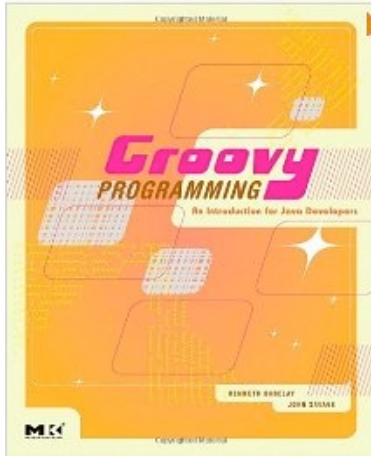
http://groovyconsole.appspot.com/

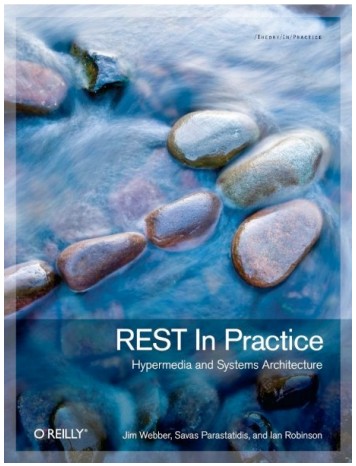


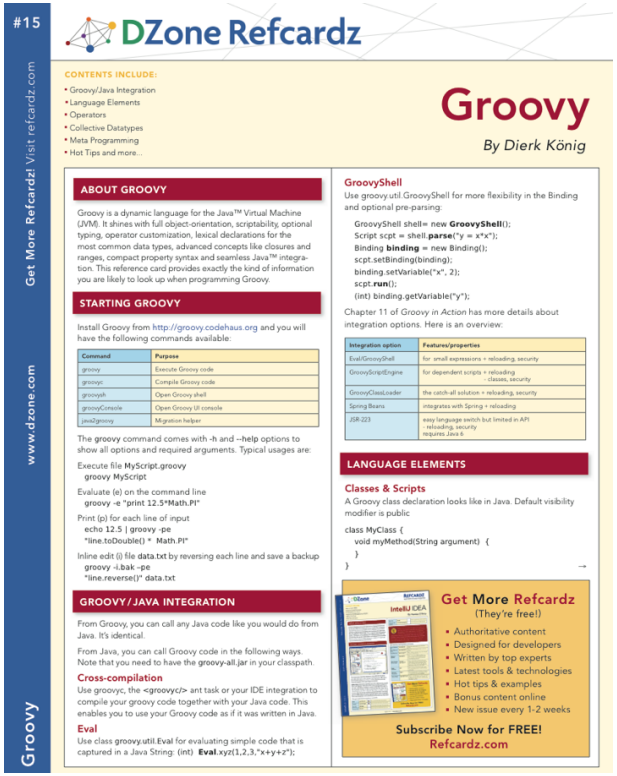
# References

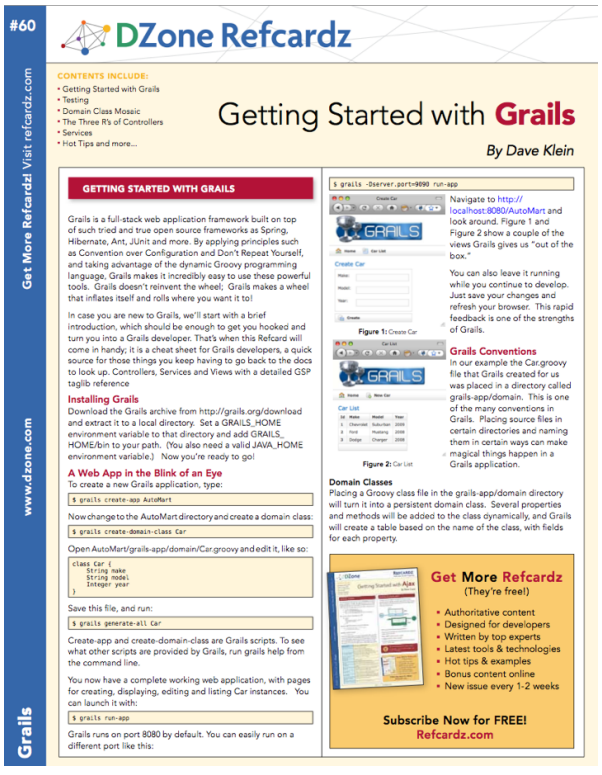

THE EXPERT'S VOICE® IN OPEN SOURCE
Beginning
Groovy
and Grails
From Novice to Professional
A practical tutorial to agile web development for the Java platform using Groovy and Grails.
Christopher M. Judd,
Joseph Faisal Nusairat,
and James Shingler
Foreword by Graeme Rocher, Grails Project Lead
Apress®


The Definitive Guide to
Grails 2
Jeff Scott Brown and Graeme Rocher
Apress®


Groovy
PROGRAMMING
An Introduction for Java Developers


REST In Practice
Hypermedia and Systems Architecture
O'REILLY®
Jim Webber, Savas Parastatidis, and Ian Robinson


#15
DZone Refcardz
Groovy
By Dierk König
CONTENTS INCLUDE:
• Groovy/Java Integration
• Language Elements
• Operators
• Collective Datatypes
• Meta Programming
• Hot Tips and more...
ABOUT GROOVY
STARTING GROOVY
GROOVY / JAVA INTEGRATION
GroovyShell
LANGUAGE ELEMENTS
Classes & Scripts
Get More Refcardz
(They're free!)
Authoritative content
Designed for developers
Written by top experts
Latest tools & technologies
Hot tips & examples
Bonus content online
New issue every 1-2 weeks
Subscribe Now for FREE!
Refcardz.com
Get More Refcardz! Visit refcardz.com
www.dzone.com
Groovy


#60
DZone Refcardz
Getting Started with Grails
By Dave Klein
CONTENTS INCLUDE:
• Getting Started with Grails
• Testing
• Domain Class Mosaic
• The Three R's of Controllers
• Services
• Hot Tips and more...
GETTING STARTED WITH GRAILS
Installing Grails
A Web App in the Blink of an Eye
Domain Classes
Grails Conventions
Get More Refcardz
(They're free!)
Authoritative content
Designed for developers
Written by top experts
Latest tools & technologies
Hot tips & examples
Bonus content online
New issue every 1-2 weeks
Subscribe Now for FREE!
Refcardz.com
Get More Refcardz! Visit refcardz.com
www.dzone.com
Grails

# Free Training Videos

http://www.skillbuilders.com/Groovy-and-Grails/Groovy-and-Grails-Training.cfm?tab=groovy-and-grails-training