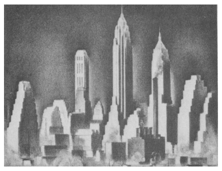
Downtown Skyline Datashop
info@DTSLData.shop
+1 347 974 3766



Cormorant/AWSHosts™

Instance IDs as hostnames in the AWS EC2 world
Peter A. Greenberg
June 30, 2020

Contents

Motivations

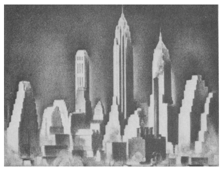
The main motivation is to make it easier to use your own, off-cloud home or office computer to manage instances that you have launched on the Amazon EC2 cloud. It can also enable you to surf the web more anonymously.

What it is

Cormorant/AWSHosts is an Amazon Machine Image (AMI) based on Amazon Linux. As an AMI, it is only available on the Amazon EC2 cloud, and it exists to make managing cloud instances easier. What makes it special is that it enables users to refer to machines by their Amazon EC2 instance ID, rather than just by its private and public Amazon host names. On an instance launched from the Cormorant/AWS Hosts AMI, you can substitute an Amazon instance ID and it will map to the private Amazon IP4 address. You can also use the “Name” tag associated with an instance. These show up in the Amazon EC2 web console and can be arbitrarily manipulated by a user with administrative privilege.

How it does it

Cormorant/AWSHosts is pretty simple. Starting in the boot phase, the awshosts Linux service queries the Amazon EC2 service for a list of running instances in the same EC2 region as the instance it is running on. It uses this to modify the instances /etc/hosts file. This file is consulted even before DNS when a hostname needs to be resolved on the instance (see Linux manual page for nsswitch.conf and the actual file /etc/nsswitch.conf). It makes the instance ID of each running instance in the region a



hostname and it maps it to the private IP4 address associated with the instance. Finally it also makes any "Name" tag an alias for that host. You can still refer to the DNS-based private and public hostnames maintained by Amazon, too.

The Problems It Solves

Easier and More Secure Cloud Management

Presume I am a software developer or tester and I need to regularly browse to a particular server on the Amazon EC2 cloud. To do this from my home or office computer I ask that TCP port 443 (the "https" port) be opened on that machine. I access the AWS EC2 portal to find the public IP address of that instance (or equivalently, its public hostname), and I plug that into my browser.

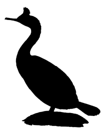
So far, not bad. But there are issues, particularly when we try to scale this approach to several web server instances. First, to save money, we refrain from using constant ("Elastic IP") addresses for these servers, and stop them each evening. Each time we stop a machine we relinquish its public IP – and we get a new one when we restart it. Thus, as a developer working on several servers, I always have my hands on that AWS EC2 console – if only to get the public IP to cut and paste into a browser. It's tedious and error prone. And there is also a potential security concern in having not one but several ports open to the public.

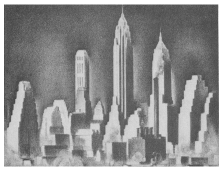
Now I learn about "port forwarding", which is a form of "tunneling" and a long-standing part of the SSH protocol. SSH is implemented by the popular (and free) "putty" and "ssh" tools, which are already used to administer nearly all Linux (and perhaps a few Windows) instances on the EC2 cloud. With port forwarding, I use an SSH client (generally either "putty" or "ssh") to first connect to one cloud server with a publicly exposed SSH port (port 22, usually). This connection, which is encrypted and authenticated through private key encryption mechanisms, is usually used as a way to communicate with a remote shell, but the protocol supports an arbitrary number of side channels (or tunnels) that share that connection. I can then tell the client to associate a port on the local machine with a port on a remote machine. In brief, a URL like http://localhost:3000 might actually take me to http://ip-172-31-62-101.ec2.internal:80 on the cloud.

That is an advance in a number of ways. First, "ip-172-31-62-101.ec2.internal" is an EC2 private hostname, which is conserved between stops and starts. It rarely changes while an instance exists. So I won't have to modify the mappings frequently. Furthermore, the ip-172-31-62-101.ec2.internal host need not have a port open to the world. Its firewall rules might restrict access to other members of its "security group", which are all cloud instances in the same AWS account and region. The single gatekeeper host assures that everyone coming in has a certain well-guarded private key (up to 1024 bits long), which is a better form of authentication than passwords. So port forwarding is clearly an advance, perhaps with modest tradeoffs in performance as bytes make an additional hop.

But still, there are issues. Instances come and go. Private hostnames are not mnemonic. Let's say our "ip-172-31-62-101.ec2.internal" machine were functionally replaced by a different instance. Now I have to go in to the "putty" port forwarding screen and find the old name and replace it with the new one. That is not fun or easy. There's still a lot of bookkeeping.

And that is where Cormorant/AWSHosts comes in. On a server instance launched from the Cormorant/AWSHosts AMI, AWS instance ids are valid hostnames that are associated with the private IP





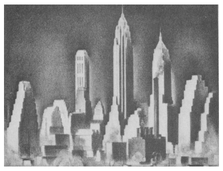
addresses of their instances. Further, the Name tags attached to instances, which are mnemonics created by humans, are also valid alias hostnames for those instances. These hostnames come and go automatically as instances are started (launched), stopped, and terminated, and as users enter Name tags, delete them, and move them between instances. It is as if the AWS EC2 console has become a host naming tool. This makes that Cormorant/AWSHosts server instance an ideal gatekeeper to use with port forwarding. To denote the server side of each port forwarding spec, you can give a Name tag, which can be arbitrarily moved around from instance to instance, at will. Or use an AWS instance ID or a private IP address. Whatever you do, you'll be assured of an encrypted, authenticated connection.

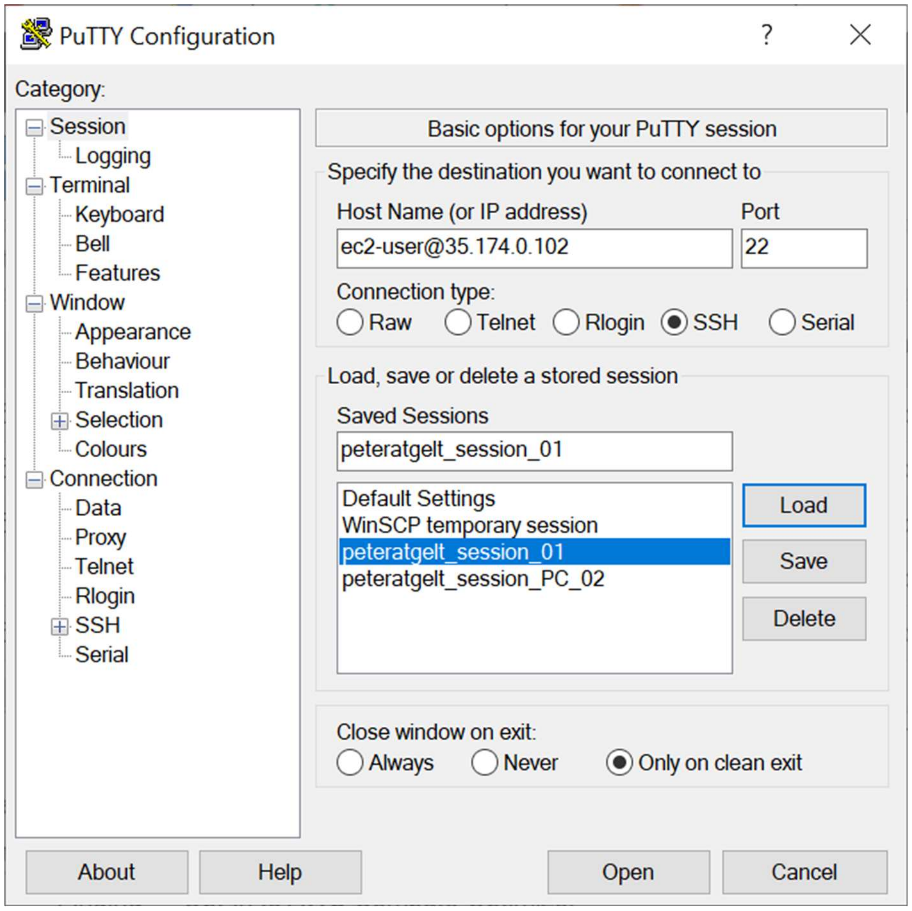
Additionally, the Cormorant/AWSHosts server instance comes with an Apache httpd (i.e. web) server running on it. This includes a "forward proxy". This enables the use of natural URLs that are based on instance Name tags, instance IDs, or private IP addresses. Traffic to the proxy is port forwarded through the authenticated and encrypted channel.

Walkthrough

Lets work through an example. We will use "putty" to connect to a Cormorant/AWSHosts server instance (i.e. an instance launched from the Cormorant/AWSHosts AMI). Then we will create a few port forwards and demonstrate how to use them.

First, the establishment of a nice secure channel. What is not shown for lack of space is the "keypair" that is used to encrypt the channel and authenticate the client to the server. We press the "Open" button and we get in. Putty opens a new (usually black-background) window in which we have a shell session on the remote Cormorant/AWSHosts instance. Not a big deal, so far.





PuTTY Configuration

Category:

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): ec2-user@35.174.0.102
Port: 22

Connection type: Raw / Telnet / Rlogin / [✓ SSH] / Serial

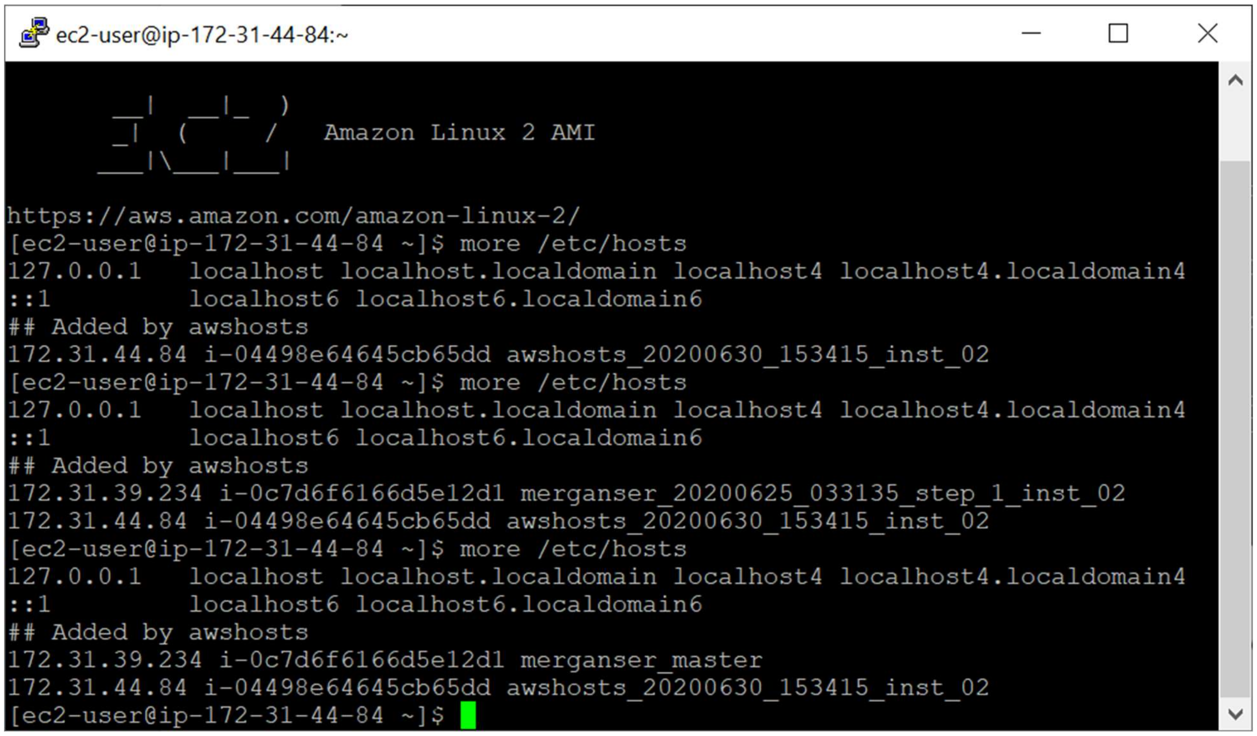
Load, save or delete a stored session

Saved Sessions: peteratgelt_session_01

Default Settings
WinSCP temporary session
peteratgelt_session_01
peteratgelt_session_PC_02

Close window on exit: Always / Never / [✓ Only on clean exit]

Figure 1: connecting to a Cormorant/AWSHost Instance (Public IP: 35.174.0.102)

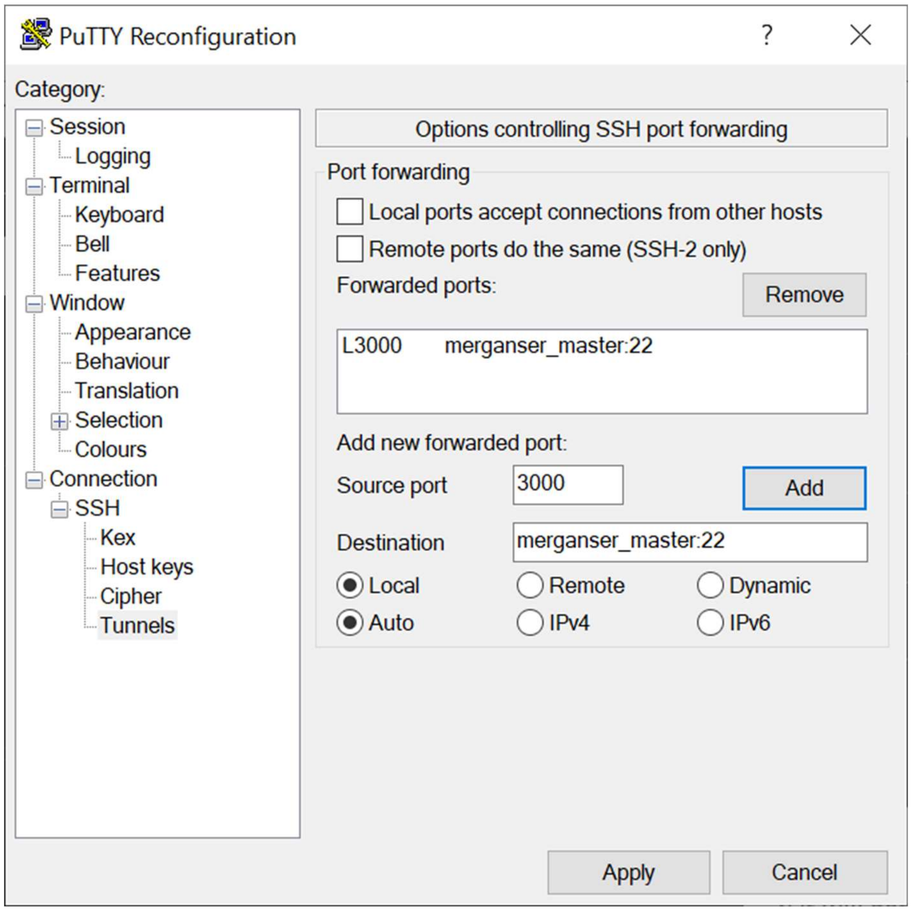

ec2-user@ip-172-31-44-84:~

```
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-44-84 ~]$ more /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost6 localhost6.localdomain6
## Added by awshosts
172.31.44.84 i-04498e64645cb65dd awshosts_20200630_153415_inst_02
[ec2-user@ip-172-31-44-84 ~]$ more /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost6 localhost6.localdomain6
## Added by awshosts
172.31.39.234 i-0c7d6f6166d5e12d1 merganser_20200625_033135_step_1_inst_02
172.31.44.84 i-04498e64645cb65dd awshosts_20200630_153415_inst_02
[ec2-user@ip-172-31-44-84 ~]$ more /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost6 localhost6.localdomain6
## Added by awshosts
172.31.39.234 i-0c7d6f6166d5e12d1 merganser_master
172.31.44.84 i-04498e64645cb65dd awshosts_20200630_153415_inst_02
[ec2-user@ip-172-31-44-84 ~]$
```

Figure 2: The login shell. Note that /etc/hosts has instance IDs related to private IP4s

So we use the shell to see the /etc/hosts file, and we see it looks a little unusual. It has entries for the two instances running in this region in this account. It's already interesting, but it's about to get more so as we add port forwarding rules.



[ ] Local ports accept connections from other hosts
[ ] Remote ports do the same (SSH-2 only)

Forwarded ports:

```
L3000       merganser_master:22
```

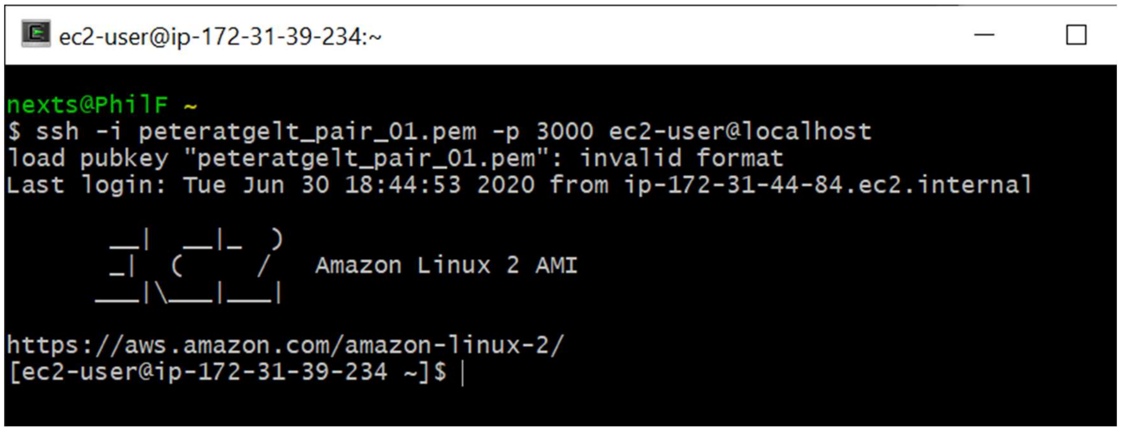
Add new forwarded port:

Source port: 3000
Destination: merganser_master:22
[✓ Local] Remote Dynamic
[✓ Auto] IPv4 IPv6

Figure 3: creating a tunnel from localhost:3000 to merganser_master:22

Here we have created a tunnel from port 3000 on the localhost to port 22 on “merganser_master”. What is “merganser_master”? It is the “Name” tag associated with a particular instance in the same region – you can see it in the listing of /etc/hosts in figure 2. Thus it is an alias for a particular cloud instance. What is happening now is that besides handing the regular shell connection, “putty” is also listening on port 3000 on the local client computer. Should anything connect to it, it will tunnel a connection through to the “sshd” server on the other end, which will then connect it to port 22 on merganser_master, which it resolves according to the nsswitch.conf settings on the instance it is running on. Which tell it to look first at /etc/hosts. Which points merganser_master at the IP4 address of the cloud instance with the instance ID “i-0c7d6f6166d5e12d1”.

So now, let’s see that work. I am going to use my command line “ssh” client this time to log in to merganser_master. All the client needs to know is “localhost” and port 3000.



```
ec2-user@ip-172-31-39-234:~

nexts@PhilF ~
$ ssh -i peteratgelt_pair_01.pem -p 3000 ec2-user@localhost
load pubkey "peteratgelt_pair_01.pem": invalid format
Last login: Tue Jun 30 18:44:53 2020 from ip-172-31-44-84.ec2.internal

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-39-234 ~]$ |
```

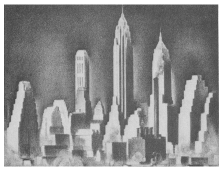
Figure 4: command-line "ssh", tunneled through putty's ssh connection

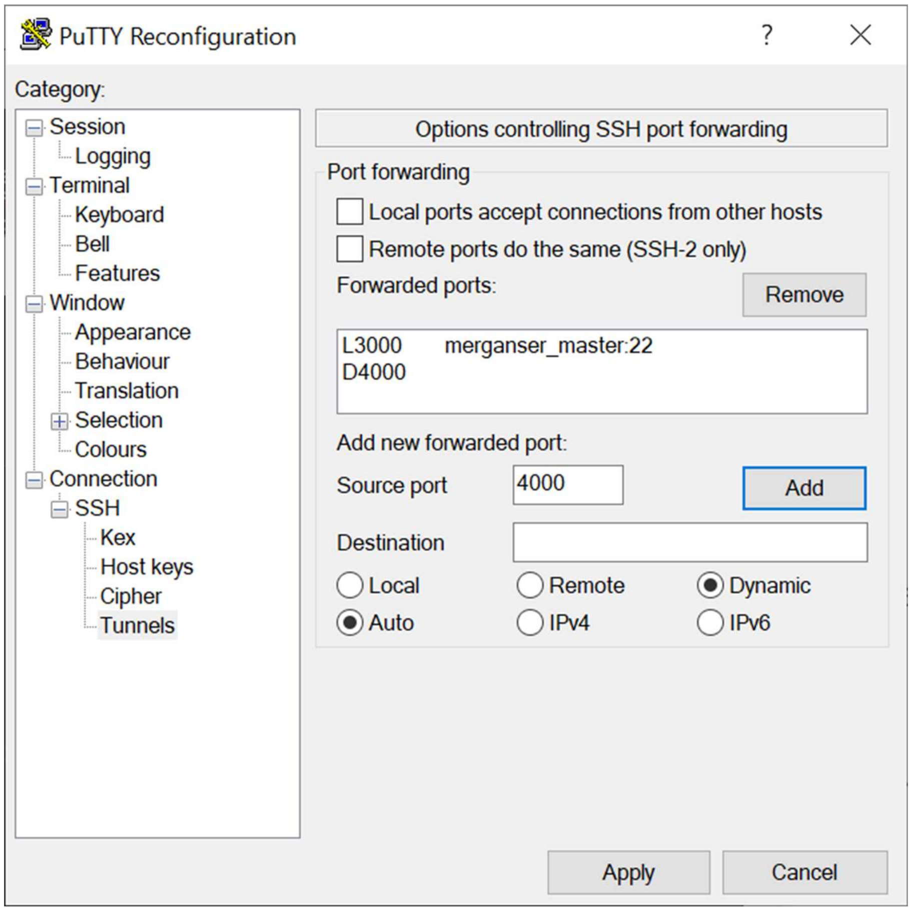
Note that all I told “ssh” to connect to port 3000 on localhost. Putty transparently connected me to the target server (“merganser_master”). Further, the following is true:

- merganser_master need not have a public IP address at all
- Tomorrow, merganser_master can might point to a different private IP (let’s say a user moved the Name tag from one instance to another), but my putty configuration need not change.

Web Proxy

The Cormorant/AWSHosts instance can also serve as an inexpensive web proxy. Here is the easiest way. Besides forwarding a particular port, “ssh” clients including putty can function as a “SOCKS5” proxy. These are very general proxies: the client contacts the proxy and asks to be connected to a particular host and port. Setting up the SOCKS5 proxy in putty is pretty easy:





Port forwarding
- [ ] Local ports accept connections from other hosts
- [ ] Remote ports do the same (SSH-2 only)

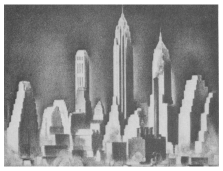
Forwarded ports:

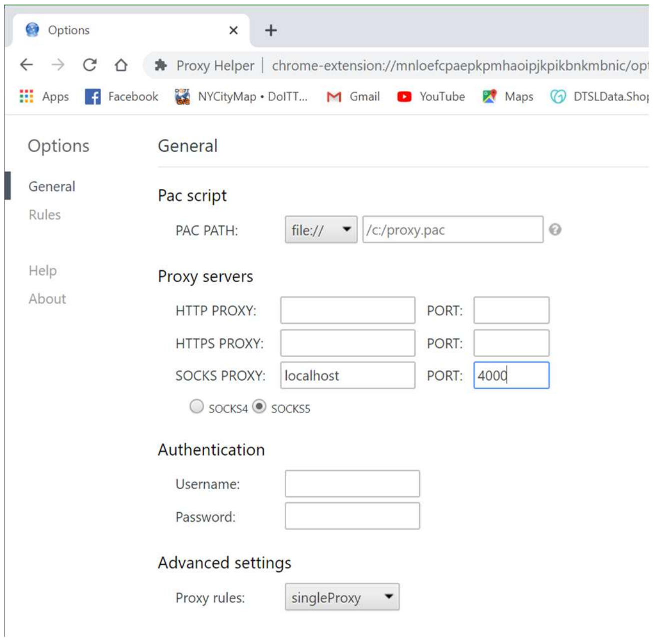
```
L3000     merganser_master:22
D4000
```

Add new forwarded port:
Source port: 4000
Destination:
Local / Remote / [✓ Dynamic]
[✓ Auto] / IPv4 / IPv6

Figure 5: added the SOCKS5 proxy, which putty calls "dynamic", on port 4000

We set up our SOCKS5 proxy to hang out on localhost port 4000. Now let's set the Google Chrome browser to use it, I use a nifty extension called "Proxy Helper" (https://chrome.google.com/webstore/detail/proxy-helper/mnloefcpaepkpmhaoipjkpikbnkmbnic) which directs Chrome to the proxy without changing other settings on my PC. Here is its configuration screen:





Options | General

Pac script

PAC PATH: file:// /c:/proxy.pac

Proxy servers

HTTP PROXY: PORT:

HTTPS PROXY: PORT:

SOCKS PROXY: localhost PORT: 4000

SOCKS4 [✓ SOCKS5]

Authentication

Username:

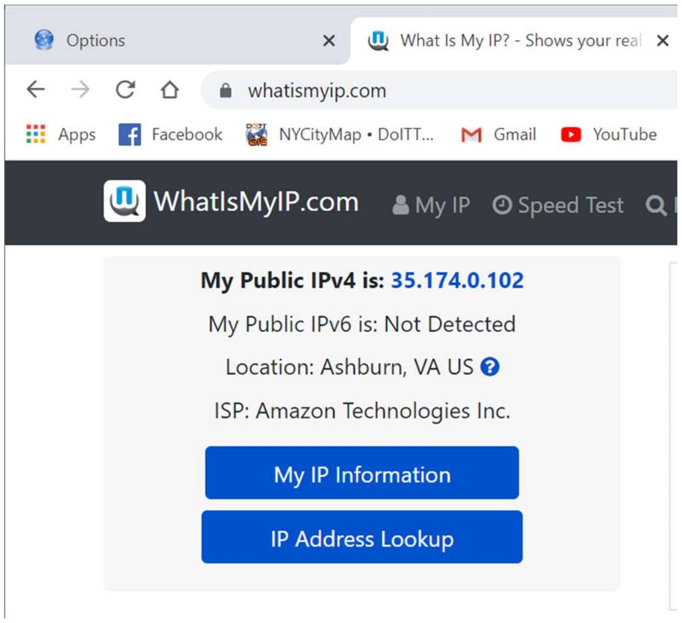
Password:

Advanced settings

Proxy rules: singleProxy

Figure 6: set browser to use putty's SOCKS5 server on port 4000

Now I go to www.whatismyip.com and sure enough, it thinks I am at Amazon:



whatismyip.com

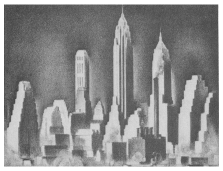
My Public IPv4 is: 35.174.0.102

My Public IPv6 is: Not Detected

Location: Ashburn, VA US

ISP: Amazon Technologies Inc.

In addition to the SOCKS5 capability, Cormorant/AWSHosts also comes with an http proxy on port 8080 of the server. Thus, one might forward port 8080 on the client to port 8080 on the server (8080 in the local port field, and localhost:8080 in the remote port – remember the hostname is relative to the server (so localhost means the server in this case). Then you tell the operating system or web browser that there is an http proxy on localhost (the client machine – argh!) port 8080. http and SOCKS5 proxies are essentially similar in action but some applications might find one easier to use than the other.



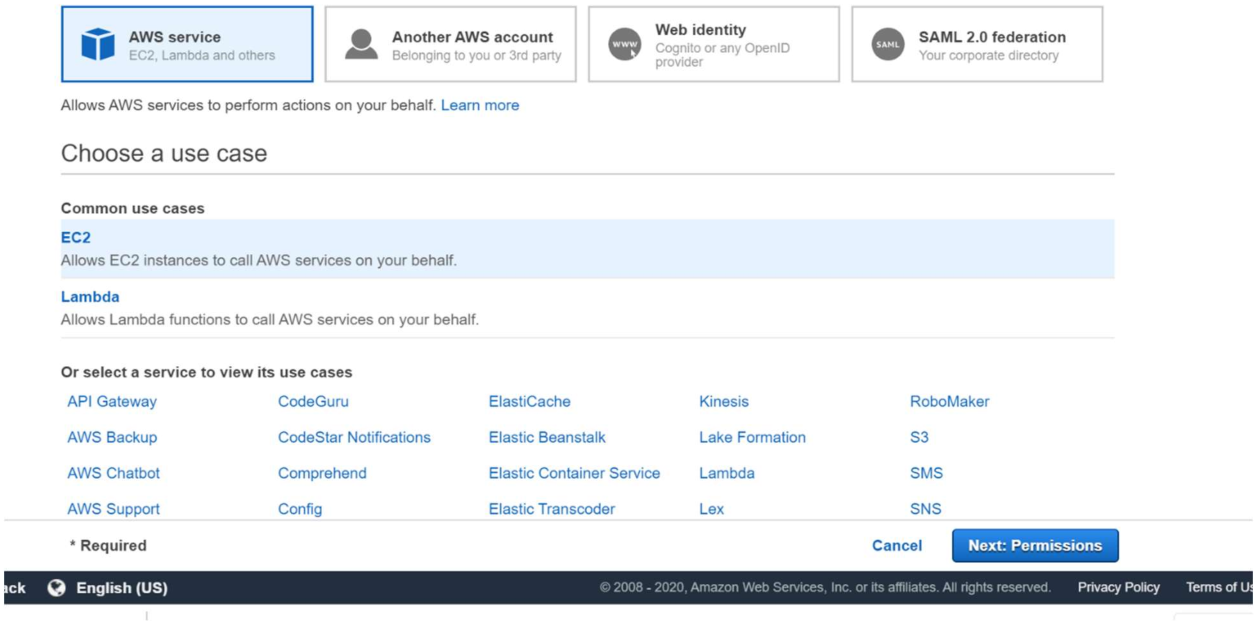
Setup

One-time only: create an IAM Role with EC2 read privileges

The Cormorant/AWSHosts software uses the AWS EC2 command-line interface (CLI) to obtain information about running instances. This requires certain privileges. The system that controls privileges in AWS is called AWS IAM. You **might** want to read up on IAM, but I will give it here cookbook-style, and it will probably just work. But if you really want to know, here some of the resources that I used:

- https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html
- https://aws.amazon.com/premiumsupport/knowledge-center/iam-role-not-in-list/

Ok so let's get started. To create your role, first go to the IAM portal: https://console.aws.amazon.com/iam . Navigate to "Roles" (note: Roles seem to be unrelated to users and groups). Click on "Create Role" button. Assure that "AWS Service" and "EC2" are highlighted. Refer to this screen for right configuration, then press "Next: Permissions" in lower right corner:



Allows AWS services to perform actions on your behalf. Learn more

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

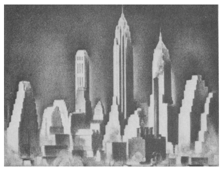
Lambda
Allows Lambda functions to call AWS services on your behalf.

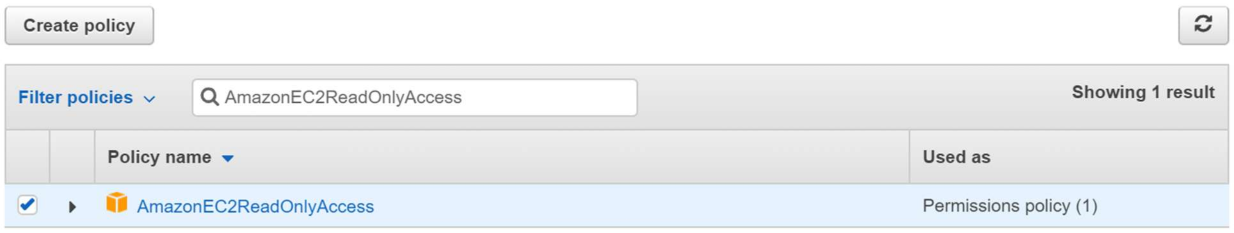
Or select a service to view its use cases

API Gateway	CodeGuru	ElastiCache	Kinesis	RoboMaker
AWS Backup	CodeStar Notifications	Elastic Beanstalk	Lake Formation	S3
AWS Chatbot	Comprehend	Elastic Container Service	Lambda	SMS
AWS Support	Config	Elastic Transcoder	Lex	SNS

* Required
Cancel
Next: Permissions
ack English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Us

On the next screen there are a zillion "policies", whatever they are. Don't worry about it. Just enter "AmazonEC2ReadOnlyAccess" in the "Filter policies" box to find the one policy with that exact name. Then check the checkbox to the left of that row (Note: clicking on the blue link instead of checking the box brings up a separate screen that can be confusing. You don't need to do that.) Refer to this screen for proper configuration. Then click "Next: Tags" in the lower right (not shown to conserve space in illustration below) to go to next screen.



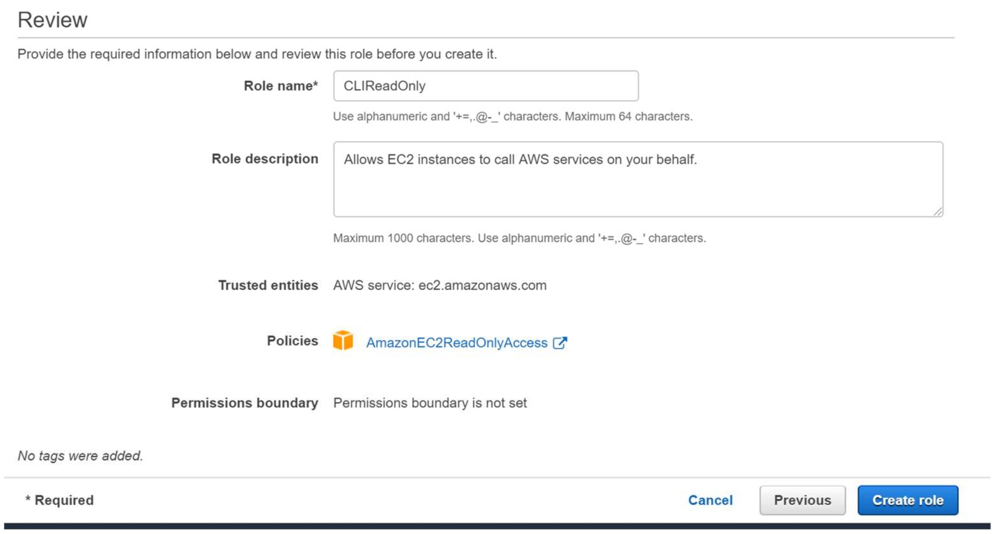


Create policy

Filter policies: AmazonEC2ReadOnlyAccess — Showing 1 result

		Policy name	Used as
✓	▸	AmazonEC2ReadOnlyAccess	Permissions policy (1)

Ignore the "Tags" screen. Simply click "Next: Review" to go to the next and last screen. You're almost done! Finally on this screen, you give your role a name. It can be anything, but you will need to remember it when you launch your instance of the Cormorant/AWSHost AMI a little later. I suggest the role name "CLIReadOnly". See the following screen for proper configuration:



Review

Provide the required information below and review this role before you create it.

Role name*	CLIReadOnly
	Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters.
Role description	Allows EC2 instances to call AWS services on your behalf.
	Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.
Trusted entities	AWS service: ec2.amazonaws.com
Policies	AmazonEC2ReadOnlyAccess
Permissions boundary	Permissions boundary is not set

No tags were added.

* Required — Cancel — Previous — Create role

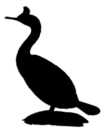
Now, finally, press "Create Role". Now you can attach this role to an instance when you launch it, and also attach it to a running instance if you forget to do it at launch time. Doing this imbues the software on that instance with the right to get a listing of your instances, as if it were you. Since you granted read-only access, the software cannot launch instances or make other changes.

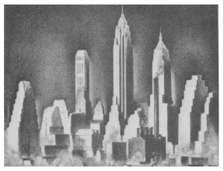
Fortunately, you won't have to do that again, at least not in this account.

Maybe more than once: launch an instance of the AMI

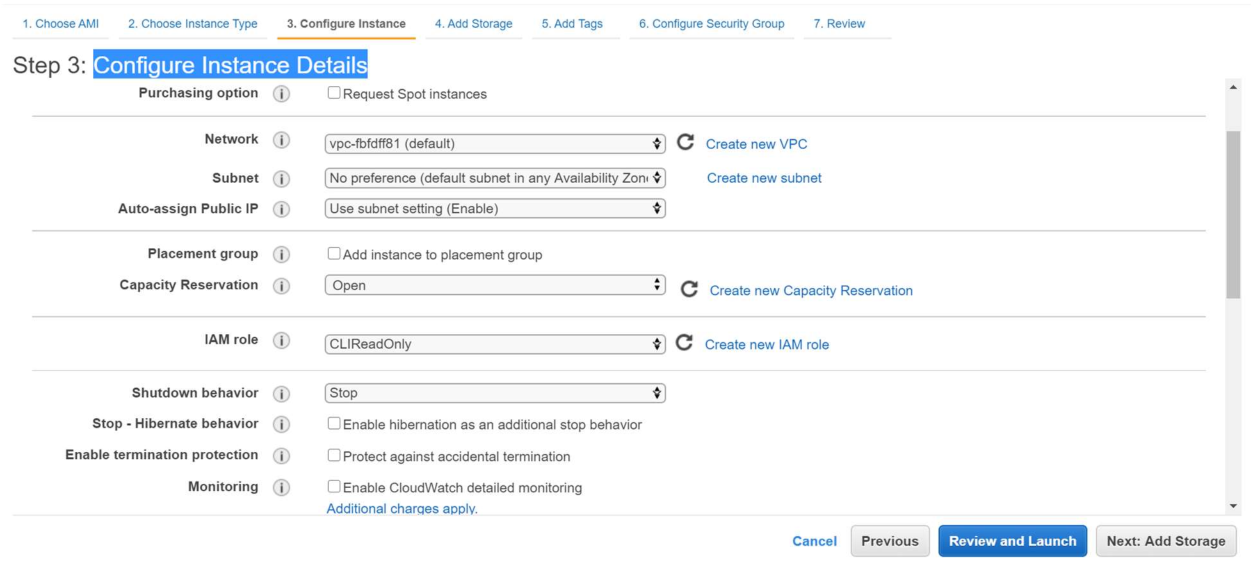
Now is the fun part where you launch an instance of the Cormorant/AWSHosts AMI. To do this, go to your account in AWS Marketplace, subscribe to the product, and then follow instructions to launch an instance. Make sure you are in the right region when you do this. If you get lost after you subscribe, you can look for "Your Software" and then launch from there.

Once you are in the launcher, you will go through the usual series of screens where you choose various parameters for your new instance. One note on the size: micro is fine. Cormorant/AWSHost is not a resource hog! Some people might need more network bandwidth, but I think micro is a great, inexpensive choice for most loads. Launch as you normally would a Linux instance, with the following notes:





One the 3rd screen, "Configure Instance Details", scroll down to the middle, and choose the IAM Role you created above (the one I suggested you call "CLIReadOnly"). It should appear on the drop-down menu in that field. Whoohoo, this is where all that hard work comes in! (To review, setting this field gives the software on that instance the right to query Amazon service for a list of running instances and details relating to them, all without getting further credentials from you.) The other fields are up to you, but if you don't understand them, just leave the defaults. See this screen for suggested conformation:



Step 3: Configure Instance Details

Purchasing option: ☐ Request Spot instances

Network: vpc-fbfdff81 (default) — Create new VPC

Subnet: No preference (default subnet in any Availability Zone) — Create new subnet

Auto-assign Public IP: Use subnet setting (Enable)

Placement group: ☐ Add instance to placement group

Capacity Reservation: Open — Create new Capacity Reservation

IAM role: CLIReadOnly — Create new IAM role

Shutdown behavior: Stop

Stop - Hibernate behavior: ☐ Enable hibernation as an additional stop behavior

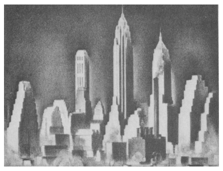
Enable termination protection: ☐ Protect against accidental termination

Monitoring: ☐ Enable CloudWatch detailed monitoring
Additional charges apply.

One final note on security group. The only port that needs to be open to this host is TCP port 22 (the SSH port). Generally, this is left open to the outside world (the console gripes about this, but SSH is secure – see the next paragraph). If you have a stable IP address at home or work, you can limit the source IP address(es) and gain a little bit of security. You do **not** need to open any other ports on a Cormorant/AWSHosts instance. Please note that your instance has a little web server running on port 80 and a proxy server running on port 8080. Do **not** open these ports in your security group, as that could expose private information (your list of running instances) to outsiders without a password. Below, you will learn how to "tunnel" into these ports through the encrypted and authenticated SSH protocol connection. In this manner, *only* people who have securely tunneled in through SSH will be able to access these servers, which is what we want. Setting up tunneling through "putty" is described below.

In general, SSH is considered a very secure protocol. It always uses TLS encryption end-to-end (the same kind of encryption that is used on the web with "https" URLs), and is thus hardened to man-in-the-middle and other attacks. It also provides a strong degree of authentication: you need a large "private key" to get in. This must match a "public key" stored by Amazon and dropped into your instance at launch time. Keep your private key safe – it is essentially a password. Ideally it should never move from the place where you originally created it.

Finally, you will get to the screen where you say yeah, launch my instance. Your console will show the screen in an "Initializing" state for up to a few minutes. Even after it shows as "Running" it may not be ready to accept connections for a minute or so. This is part of the usual Linux instance booting process (it takes a little while for sshd to be ready to accept connections) and is not specific to or even



significantly affected by Cormorant/AWSHosts software. If you have problems connecting, wait a minute or two and try again.