# Cloud Formation – CA1

GitHub: [conorh-devops/atu-iac-ca1: Repo for CA1 (github.com)](https://github.com/conorh-devops/atu-iac-ca1)

Student ID: **L00177599**

**Conclusion:**

This assignment was focused on creating a mini-network in AWS for a small business and delivering the solution using infrastructure as code. The aim was to provide a solution to the business that would spread resources across two availability zones both containing public and private subnets. In each subnet part of the application would be hosted and tied in with a suitable security group to limit access to only necessary ports and IP addresses.

I started the task by opening Cloud Formation designer and trying to design what I thought might be a suitable approach based on the figures given to us in the brief. After many attempts to design the infrastructure and generate a template, I realised that I was lacking the knowledge required to create a basic network with the required components for a VPC and subnets. I referred back to some articles on CIDR and found a Youtube video titled “IPv4, CIDR, and VPC Subnets Made Simple!”, Schachte (2018), which gave me a much clearer understanding of the networking aspect and what a VPC was. This allowed me to focus my attempt at templating out a solution by breaking up the deployment into smaller components.   
  
When creating the VPC I had to size the IP range with CIDR. Initially I had planned to use a smaller range such as 10.0.0.0/28 as 16 hosts for each subnet seemed to be more than enough to cover the requirements including the 5 reserved IPs AWS uses which were specified in the Subnets for your VPC (2022) AWS documentation online. After some initial deployments and realising I was still lacking much of the knowledge required to design and implement a working VPC solution with networking and Route Tables I decided to go with 10.0.0.0/16 for the VPC as it seemed to be commonly used across all of the official AWS documentation and in tutorials online, . This allows the VPC 65536 individual IPs which is more than enough to cover any foreseeable future scenario. Each subnet is then further split to 10.0.X.0/20. This is a Class A private IP range as defined in the RFC 1918.

Once this initial hump was overcome and I had a successful deployment of a VPC, Internet Gateway and the required association then I included the subnets contained in the first availability zone. This required adding NAT Gateways and Route Tables along with Elastic IPs and some additional resources to complete the network infrastructure to allow communication in and out of the public subnet. The private subnet, as I discovered in Subnets for your VPC, is essentially the same resource as the public with the exception of there being no direct access to the internet from the allocated IP. In a public subnet there is a Public IP allocated and this is used to communicate directly with external IPs through an internet gateway, a private subnet can be open to the outside internet but relies on network address translation to map requests in and out of the network and does not use and internet gateway.

At this point most properties were hardcoded into the template and a deployment was tested. All appeared okay but there was no way to verify the network without going on to the next steps and deploying an EC2 instance and the attached security group.   
  
To create an instance that would be correctly secured it also required an individual security group. The reason for this is that because we have a different requirement for each instance and we need a custom security group for each one. If we had two instances that shared the same requirements then we could have a single security group that would be associated to both instances or multiple instances if required. The initial instance I set up was the jumpbox as it appeared to be a key resource for accessing all of the other resources on the VPC and verifying the work done to create the network. The jumpbox was created with a standard set of properties for a t2.micro instance and associated with the security group that allowed SSH access over port 22 in and out. Another thing of note was that when creating the EC2 instance it does give an option to associate the login with a SSH key pair. I decided to set this up manually in this instance as I wanted to download the PEM file locally for later use, although this looks to be possible from the template and there is a resource identifier for it in the documentation. In a full secured production system, I would most likely include this in the template and use output references to pass them in to the respective instances. I would also not recommend sharing the same key pair across all instances but I done it here for simplicity.   
  
The next part was to create a private instance which is the backend in availability zone one. I did note there seemed to be conflicting info in the final diagram architecture we were provided but I was unsure if this was my own misinterpretation. I implemented it as Jumpbox public, Backend private, AZ1 and the Frontend public, Database private, AZ2 as this made sense to the solution we were asked to implement and a frontend would need to be accessible by users. The private instance was similar to the public but with the relevant properties changed to use the private subnet and slightly different ingress and egress rules in security group.   
  
Once all this was deployed successfully then I was able to begin testing SSH access on the system. To do this I used the Connect option on my instances to retrieve the addresses I could SSH too. I have SSH on powershell so I used the previously downloaded PEM cert to open a connection to my jumpbox. Initially I had issues because of permissions on the file meaning AWS denied my login. This is chmod 400 on linux but I had to use Icacls.exe in windows to set similar permission before then gaining access to the jumpbox over SSH.   
  
To SSH on to the backend I needed to SSH on to the jumpbox and then SSH again from there to the private IP of the backend. To do this I had to use scp to copy the cert on to the disk of the jumpbox so I could use it when connected. The scp command done this and again I had to run chmod 400 on the jumpbox to fix permissions but once this was done, I had SSH access as expected into the backend instance. This was a milestone for me considering how much I had struggled initially and to complete the solution I just duplicated the resources for the first availability zone in the template and changed the details accordingly.

The subnet and associated resources were copied and pasted with the identifiers and properties changed to suit. The instances and security groups were slightly more involved as the security groups were different here again but not overly complicated. Once all this was done again, I revised the template and parameterised a lot of the properties I had previously hardcoded. This cleaned up the template fairly well and allowed some more control over the parameters it runs with.

Once a final successful deployment was done then I done all the required tests and took screenshots of the results for the appendix before documenting the conclusion.

In summary of the overall solution, I believe it is an okay attempt to deploy a solution using infrastructure as code and it is somewhat proven as repeatable as it was run multiple times during testing. It is quick to deploy and it passes without error using the default parameters. If I had more time to refactor the solution, I would add in better validation on the input to stop any potential issues with bad data being passed in to the parameters when creating a stack. There is a lot of room for improvement by further templating the script using nested stacks and this does seem very achievable for certain parts especially around the subnetting as it shares much of the same resources with slightly different configurations. Using output references as well would allow you to easily link another resource in the parent or even in another nested template. As something that could be shared across a team who might maintain this solution it would be a good starting point and does eliminate the need to manually provision resources for the most part (SSH key pair here). There also appears to be metrics on config drift in AWS which is interesting as it is something I have seen in other solutions were changes are made to resources manually post deployment and not updated in the deployment template. This causes issues in future deployments where new configurations can be lost or resources downgraded. Using infrastructure as code is a good way to tackle this and metrics like config drift enhance that further.   
  
At a high level as a solution for the business and their use case I would say it is near the mark as it has no actual database or frontend served but this would not be difficult to extend on the current infrastructure. A MySQL database and something to serve a webpage would be sufficient on the existing resources after some further configuration to the templates to deploy the required security groups and resources.

The security of the resources is good and does employ a block all but allow necessary approach rather than being too open to external actors. The points noted around the shared SSH key pair would improve this further and considerations would need to be made if extending the solution further in terms of database access etc. I don’t have a fixed IP but another option is to restrict the jumpbox to a set IP or collection of known IPs if necessary. This would reject attempts to connect from unknown IP addresses and greatly reduce the attack vector if the SSH key was lost.

The last point on reliability and high availability I think is okay but I don’t understand exactly why the solution is split into four different instances. If I was doing disaster recovery or high availability for a solution then it would be replicating the same resources across zones so that if one goes down then the other zone would be the failover. In this case it looks like you will either lose the backend and jumpbox or the frontend and database. I thought it might just be to demonstrate connecting to different resources in a VPC across availability zones but there are two availability zones as described in brief so this should be sufficient for customer. There is a cost/benefit analysis with every DR/HA solution and in this case, I think we are okay.

# References

**Ryan Schachte (2018)** Subnets for your VPC - Amazon Virtual Private Cloud. Available at:

https://www.youtube.com/watch?v=z07HTSzzp3o (Accessed 15 November 2022)

**Subnets for your VPC** (2022) Available at: https://docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html (Accessed 15 November 2022)

**AWS CloudFormation VPC template** (2022) Available at: https://docs.aws.amazon.com/codebuild/latest/userguide/cloudformation-vpc-template.html (Accessed 15 November 2022)

**AWS::EC2::Instance** (2022) Available at: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html (Accessed 15 November 2022)

**AWS::EC2::SecurityGroup** (2022) Available at: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group.html (Accessed 15 November 2022)

**AWS::EC2::Route** (2022) Available at: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-route.html (Accessed 15 November 2022)

**AWS::EC2::RouteTable** (2022) Available at: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-routetable.html (Accessed 15 November 2022)

# Appendix

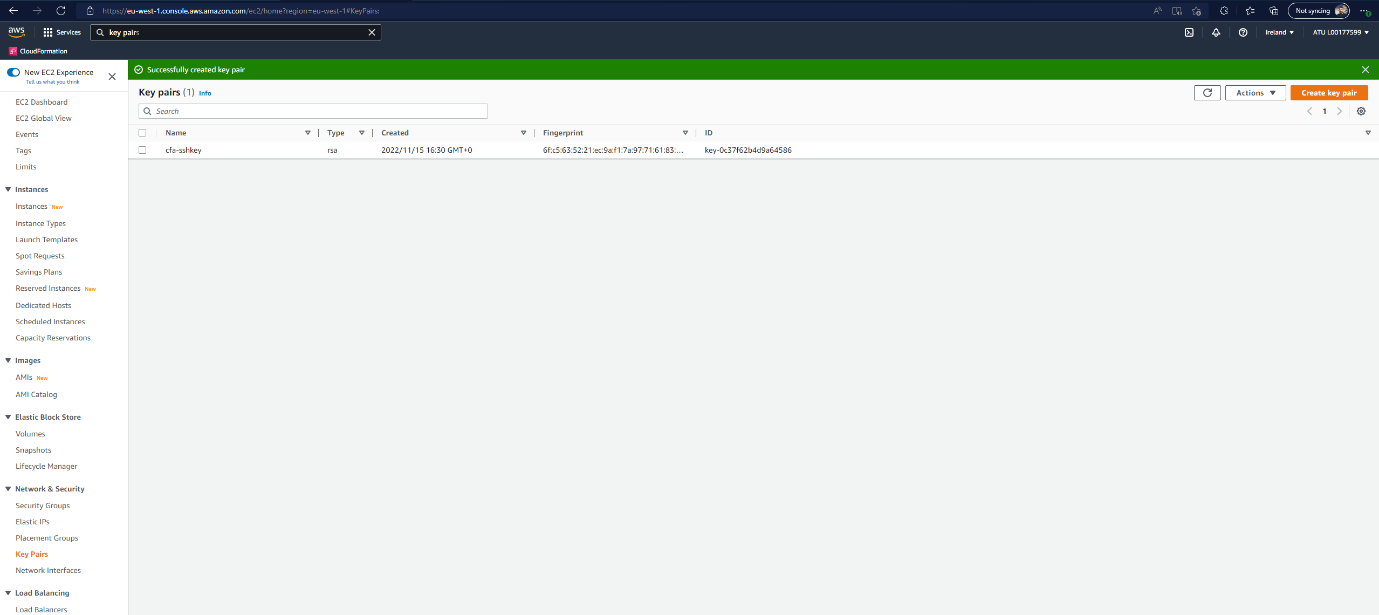


Figure Creating SSH Key Pair

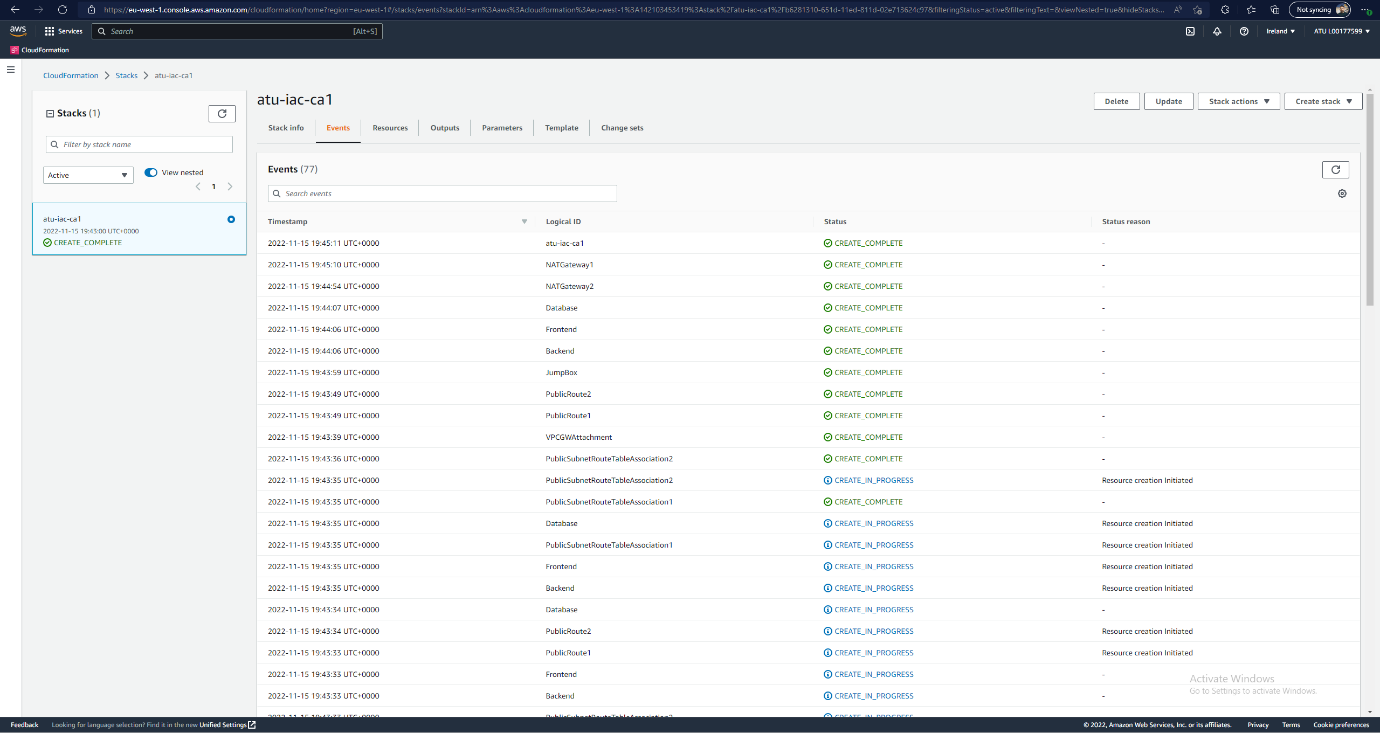


Figure Successful Deployment of Stack

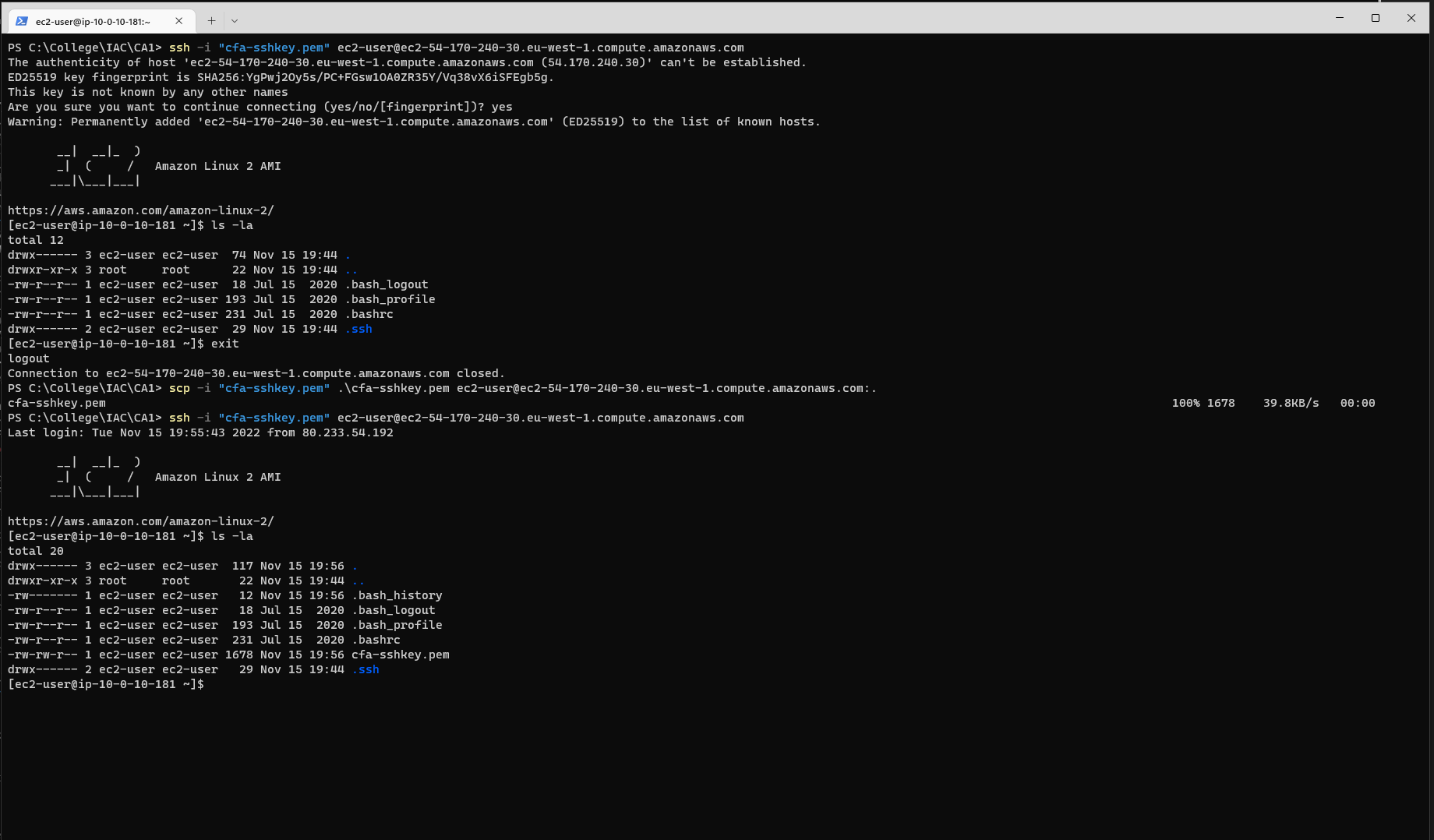


Figure Connecting to Jumpbox and Uploading Key

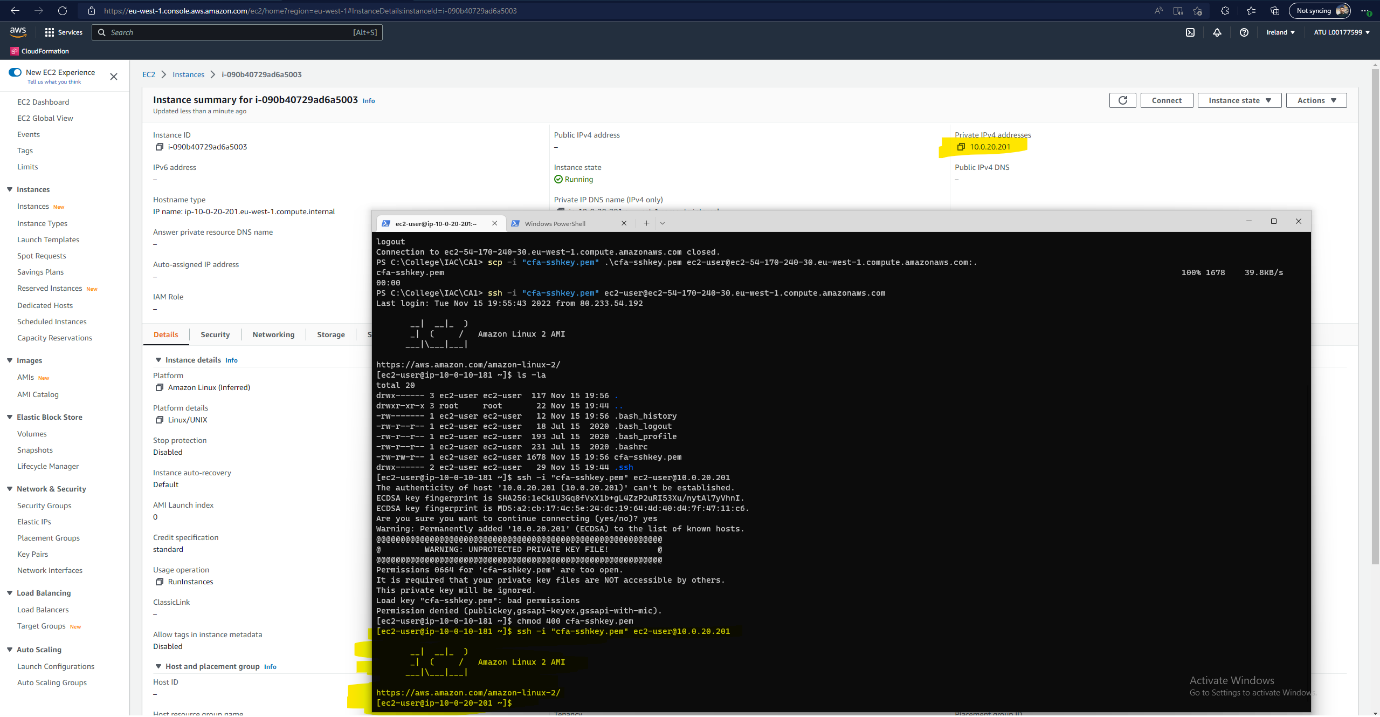


Figure SSH into Backend

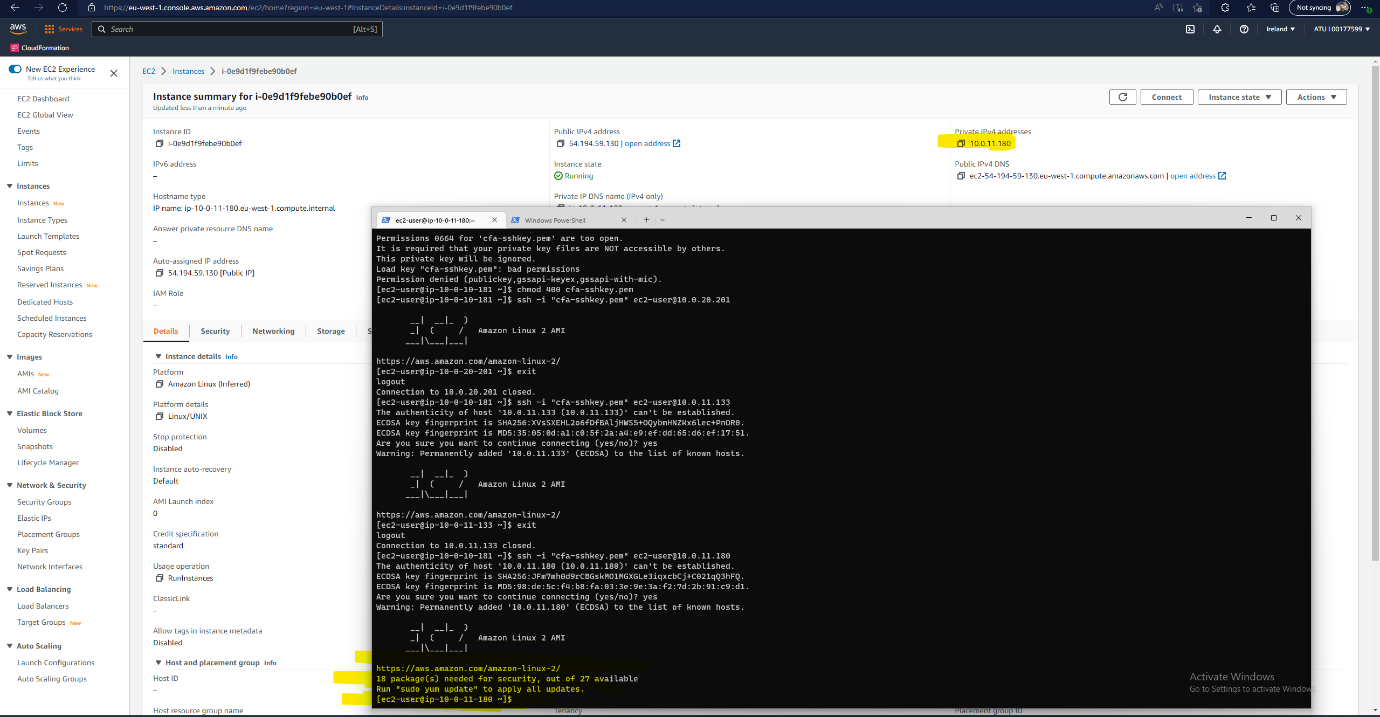


Figure SSH into Frontend

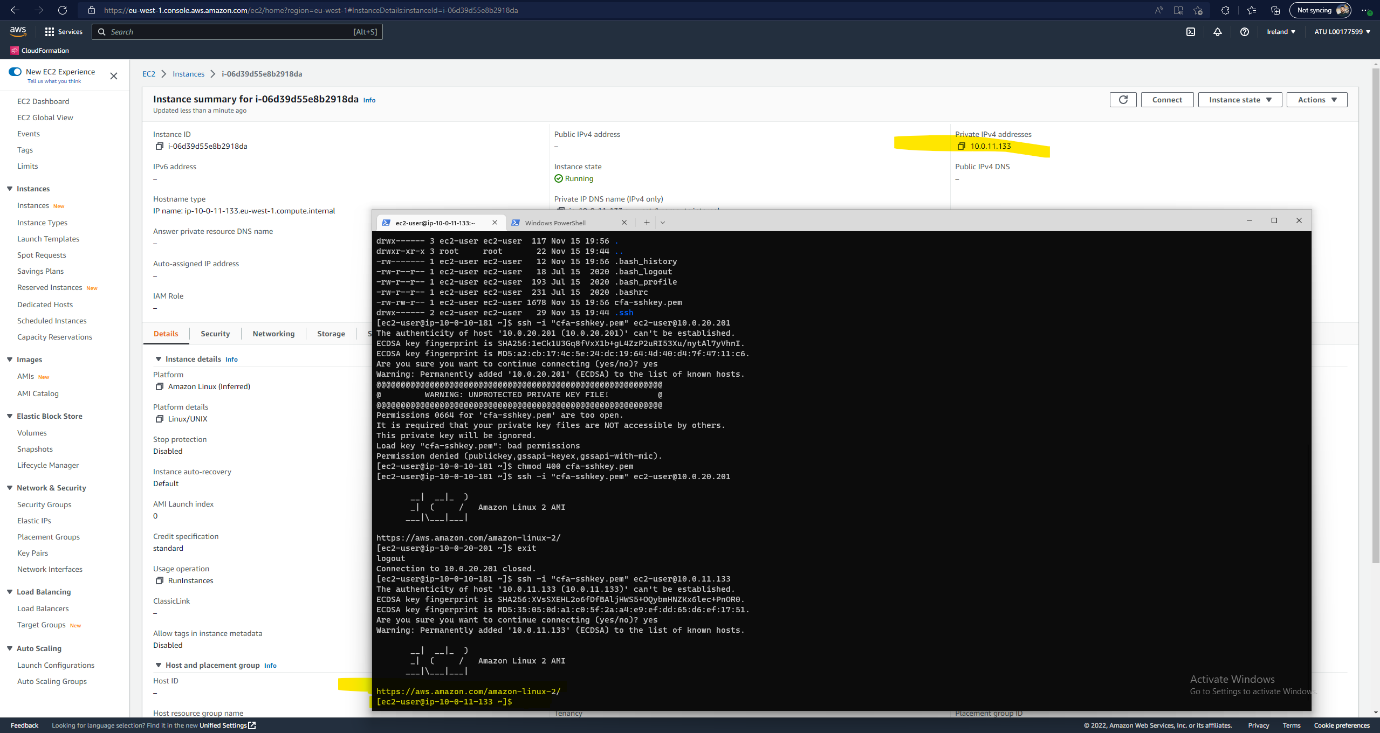


Figure SSH into Database Server

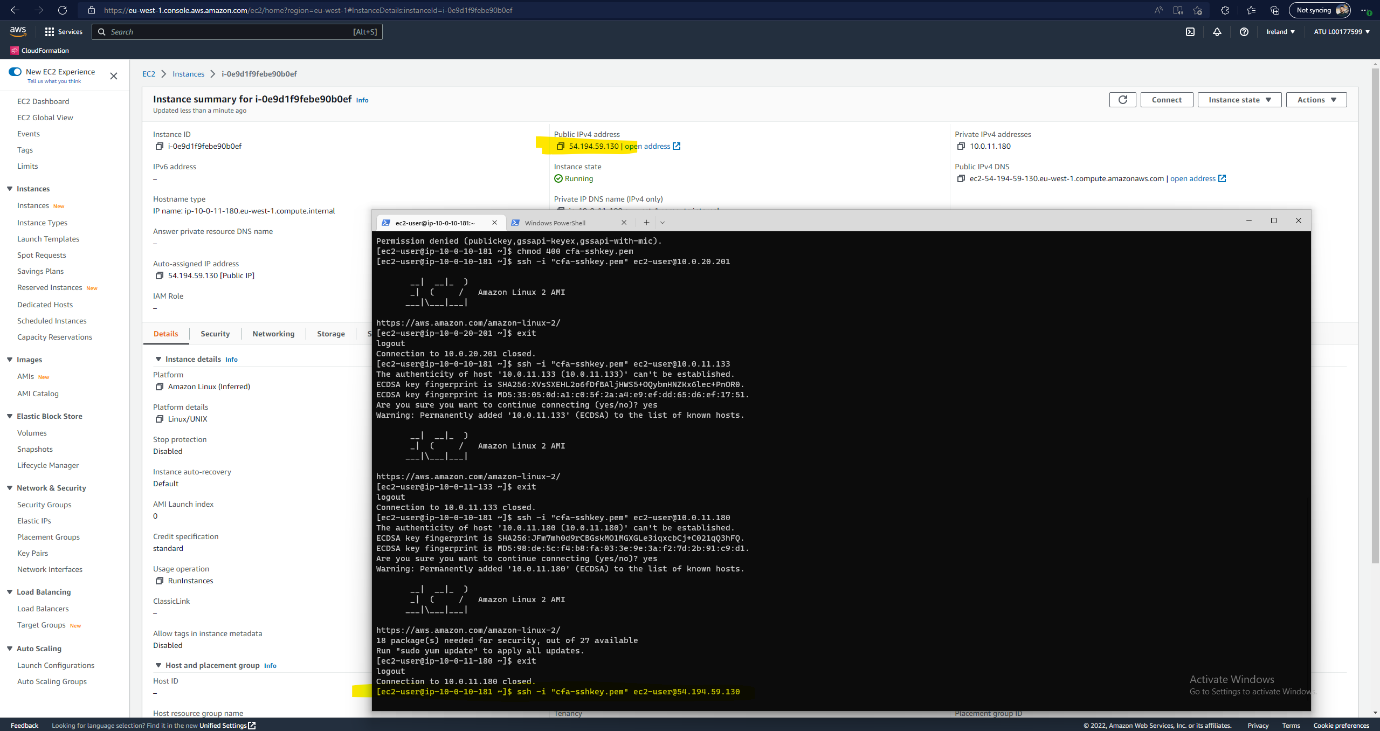


Figure Attempted SSH onto Public IP Frontend

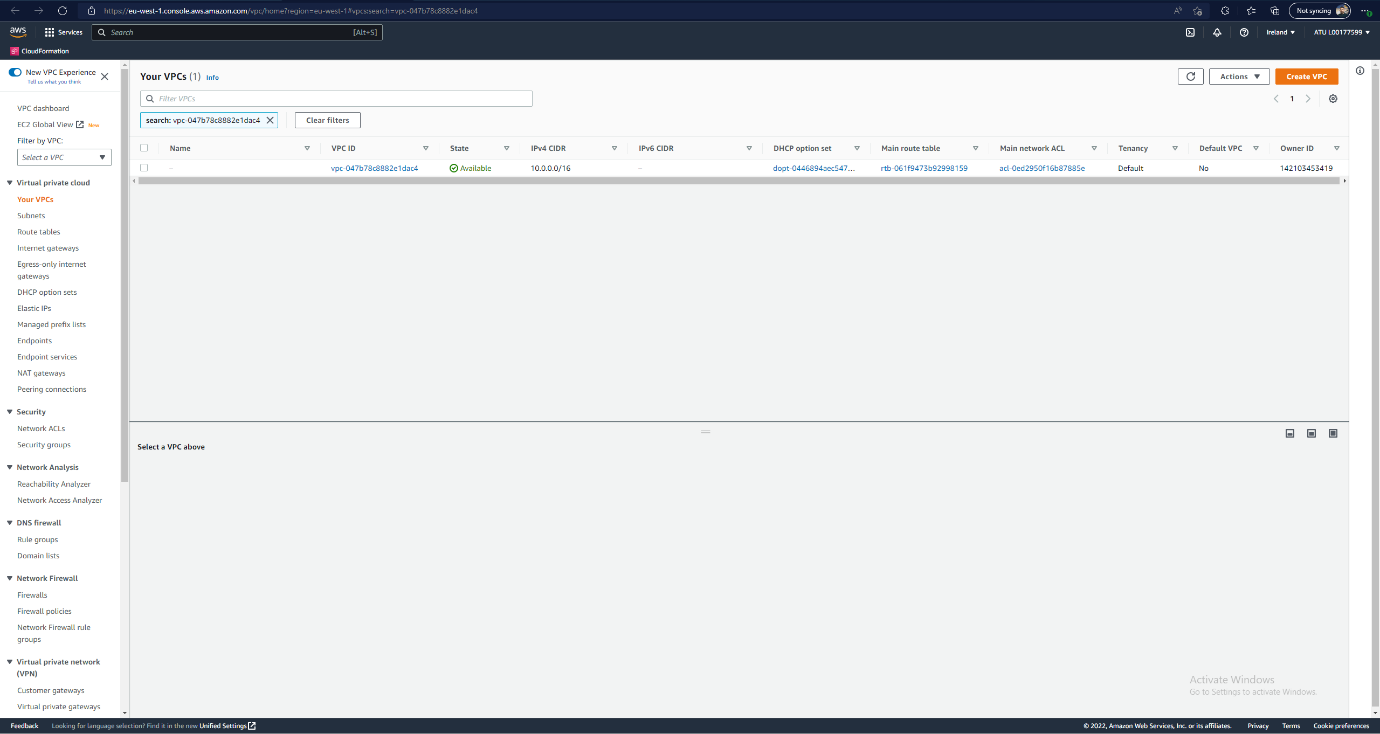


Figure VPC Console

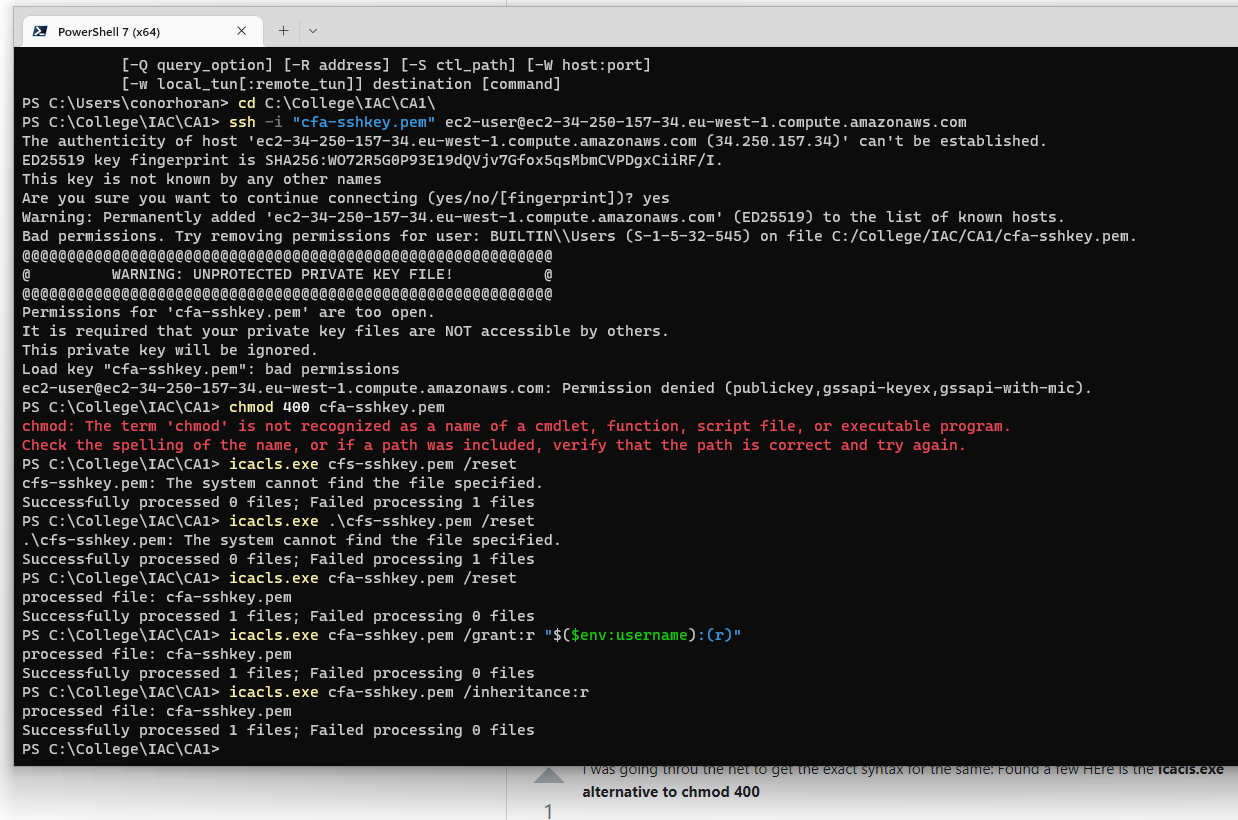


Figure Fix SSH Permissions Icacls.exe