# 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 both to provide a solution to the business that would spread resources across two availability zones, containing public and private subnets but also to explore Infrastructure as Code and how it can be used to improve the software development lifecycle in terms of consistent deployment of resources to run applications. 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.

The task was started using the Cloud Formation designer to design what might be a suitable approach based on the figures given in the brief. Research was conducted on basic networking design with the required components for a VPC and subnets. An article on CIDR, John Burke (2022), was researched and a Youtube video titled “IPv4, CIDR, and VPC Subnets Made Simple!”, Schachte (2018), gave a much clearer understanding of the networking aspect and what a VPC was. This allowed the focus to move to templating out a solution by breaking up the deployment into smaller components.   
  
When creating the VPC and sizing the IP range with CIDR initially, it was planned to use a smaller range such as 10.0.0.0/28 with 16 hosts for each subnet. This seemed to be 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 deployment testing this was changed to 10.0.0.0/16 for the VPC as it seemed to be more commonly used across all of the official AWS documentation and in tutorials online and also due to the fact additional components that were not considered initially had been discovered as requirements for the solution. This CIDR block allows 65536 individual IPs which should 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 there was a successful deployment of a VPC, Internet Gateway and the required association then the next components included were the subnets for 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 official documentation for AWS::EC2::Route and AWS::EC2::RouteTable was referenced here. The private subnet, as stated 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.

The properties were hardcoded into the template and a deployment was tested in earlier iterations and when all appeared okay the next step was to verify the network. To do this required deploying an EC2 instance and the attached security group which was done by referencing the official documentation for AWS::EC2::Instance and AWS::EC2::SecurityGroup.   
  
To create an instance that would be correctly secured it also required an individual security group. The reason for this is that because there is a different requirement for each instance each need a custom security group. If two instances shared the same requirements then a single security group could be associated to both instances or multiple instances if required. The initial instance created was the jumpbox as it was a key resource for accessing all of the other resources on the VPC and verifying the network was correct. 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 gives an option to associate the login with a SSH key pair. This was manually created in this instance to download the PEM file locally for later use. This is possible from the template and there is a resource identifier for it in the documentation. In a fully secured production system, it would be better to include this in the template and use output references to pass them in to the respective instances. It would also not be recommended to share the same key pair across all instances but it is done here for simplicity.   
  
The next resource created was the backend instance in the private subnet of the same availability zone one. It was noted here that there seemed to be conflicting info in the final diagram architecture provided but this could have been misinterpreted. The instances are implemented as Jumpbox public, Backend private, AZ1 and the Frontend public, Database private, AZ2. This made sense for the solution to be implemented and a frontend would need to be accessible by users, not private. 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, SSH access was tested on the system. To do this, the Connect option on the instances in AWS Portal was used to retrieve the addresses to SSH into. SSH was available on powershell so the previously downloaded PEM cert was used to open a connection to the jumpbox. Initially there were issues because of permissions on the file meaning AWS denied the login. This is chmod 400 on linux but on Windows Icacls.exe was used to set similar permissions before then gaining access to the jumpbox over SSH.   
  
To SSH on to the backend, first SSH on to the jumpbox and then SSH again from there to the private IP of the backend. To do this, scp was used to copy the cert on to the disk of the jumpbox so it could be used when connected. Again, chmod 400 was ran on cert on the jumpbox to fix permissions and then SSH access was gained as expected into the backend instance. This was a milestone considering how much research was required to design the solution initially. The resources for the first availability zone in the template were duplicated and details changed accordingly. This was a notable moment as it showed that using infrastructure as code that these components could be good candidates to become nested templates on further refactoring of the code.

The subnet and associated resources were copied and pasted with the identifiers and properties changed to suit and again common code across these were noted for potential to template further. The instances and security groups were slightly more involved as the security groups were different but not overly complicated. Once all this was done again, the template was revised and parameterised to remove a lot of the properties previously hardcoded. This was to increase re-usability and readability. This was noted by the fact the input could now be altered when creating the stack.

Once a final successful deployment was done and all the required tests and screenshots were taken for appendix the stack was destroyed to terminate billing.

In summary of the overall solution, there was a template created using infrastructure as code and it was proven as repeatable process to create resources. It was run multiple times during testing and consistently gave an expected result in between fixes. It is quick to deploy and it passes without error using the default parameters. If there was more time to refactor the solution then better validation on the input would be prioritised to stop bad data being passed in to the parameters when creating a stack. There is a room for improvement by further nested the stacks into smaller more re-usable templates and areas were identified were this would be possible and of benefit. Using output references would allow you to easily link another resource in the parent or in another nested template. This solution is something that could be shared across a team who could maintain it, extend it and used it to provision resources. There also appears to be metrics on config drift in AWS which is interesting as it is something sometimes not considered in IaC solutions where changes can be 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 exclusively for resource deployment and configuration 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 this is good foundation but with no actual database or frontend served currently. This would not be difficult to extend on the current infrastructure. A MySQL database and something to serve a webpage on the existing resources would fulfil the requirements with further configuration to the templates to deploy the required security groups and resources. The solution that is provided for infrastructure as code means that if more than one developer or stakeholder required it then it could be easily modified and extended on. It also will produce repeatable and predictable results due to the design in comparison to a work instruction that might require a stakeholder to follow steps to deploy the same infrastructure via the UI or console. Infrastructure as code minimises human error and with correct validation will ensure consistent results.

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. Only allowing specific IPs to connect to the jumpbox instance would further enhance security and reject attempts to connect from unknown IP addresses. This would greatly reduce the attack vector if the SSH key was lost or compromised.

The last point on reliability and high availability brought some confusion on why the solution is split into four different instances. If disaster recovery or high availability was critical for this solution then it should 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 it will either lose the backend and jumpbox or the frontend and database if one region goes down. It might just be described like this 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 a customer hosting resources in either availability zone. There is a cost/benefit analysis with every DR/HA solution and in this case it meets a good standard.

# References

**John Burke**, What is CIDR (Classless Inter-Domain Routing or supernetting)? (2022) TechTarget. Available at: https://www.techtarget.com/searchnetworking/definition/CIDR (Accessed 15 November 2022).

**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

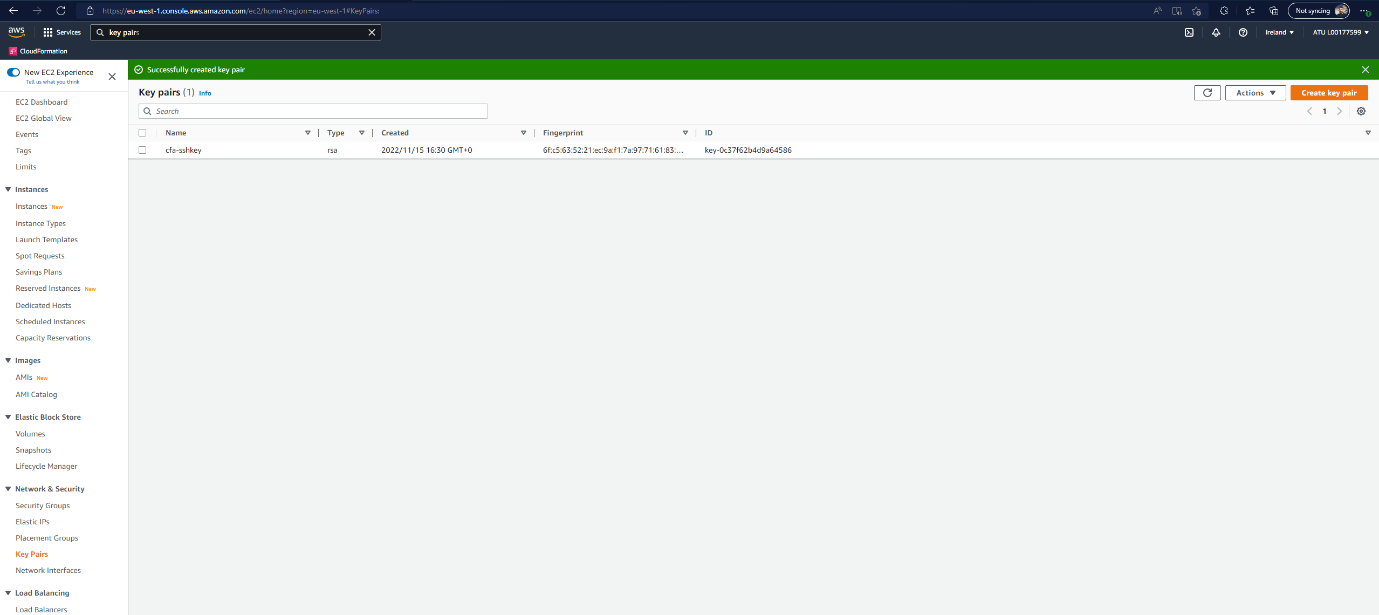


Figure Creating SSH Key Pair

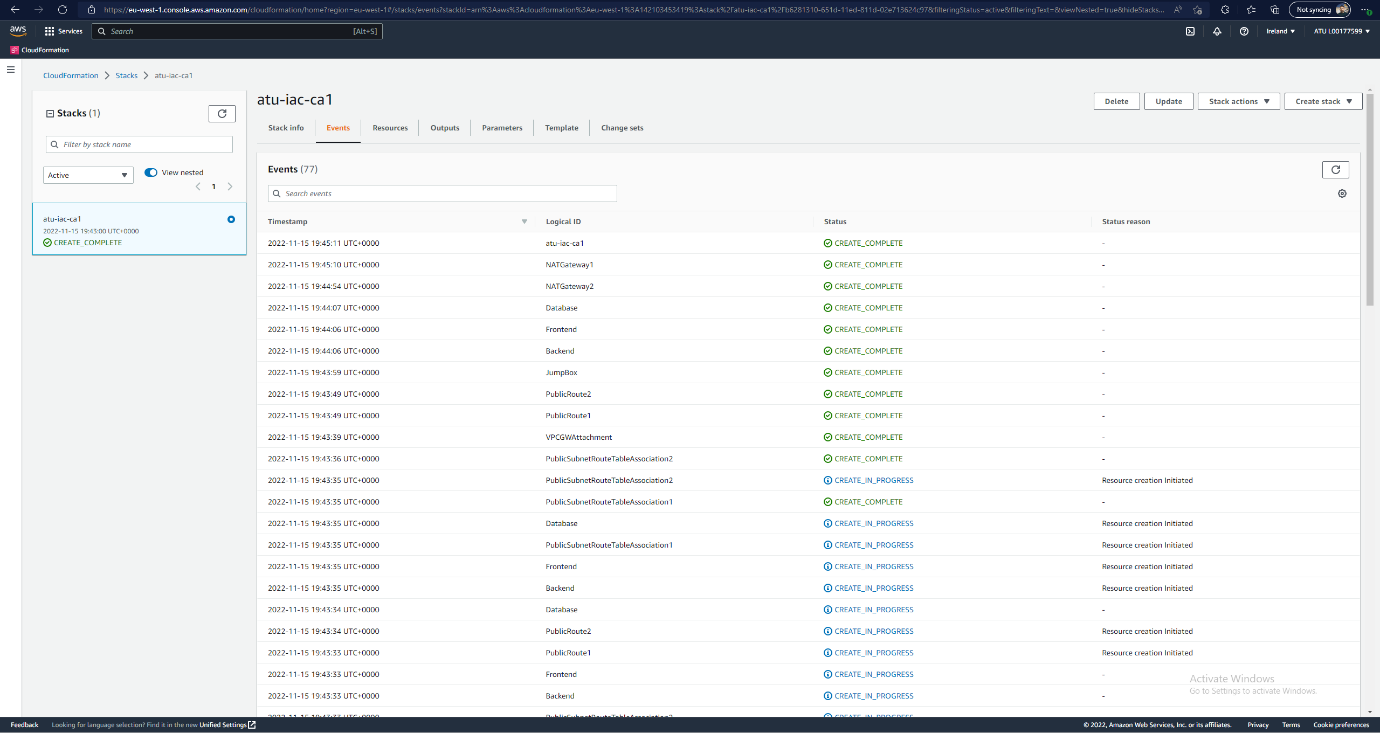


Figure Successful Deployment of Stack

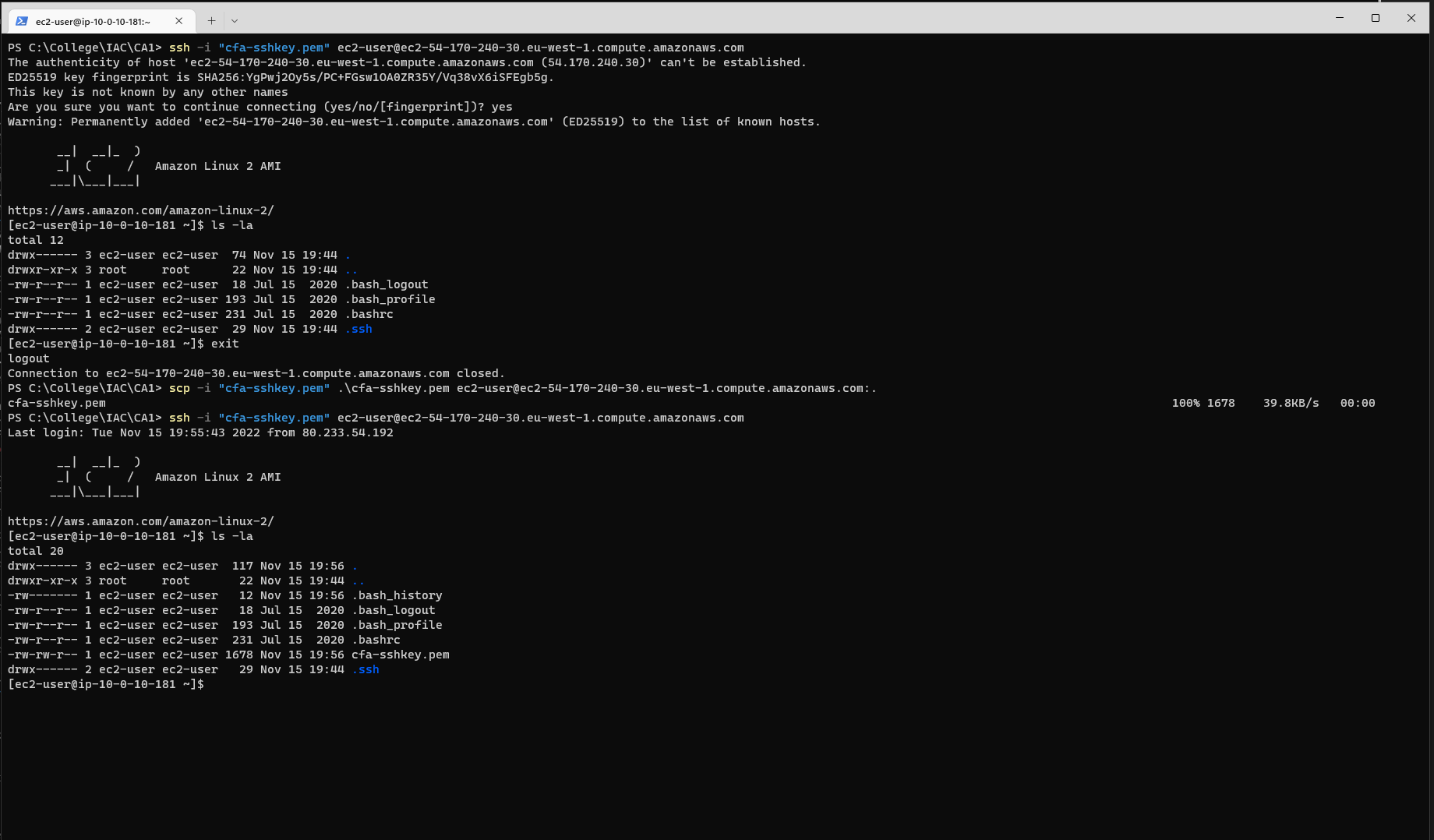


Figure Connecting to Jumpbox and Uploading Key

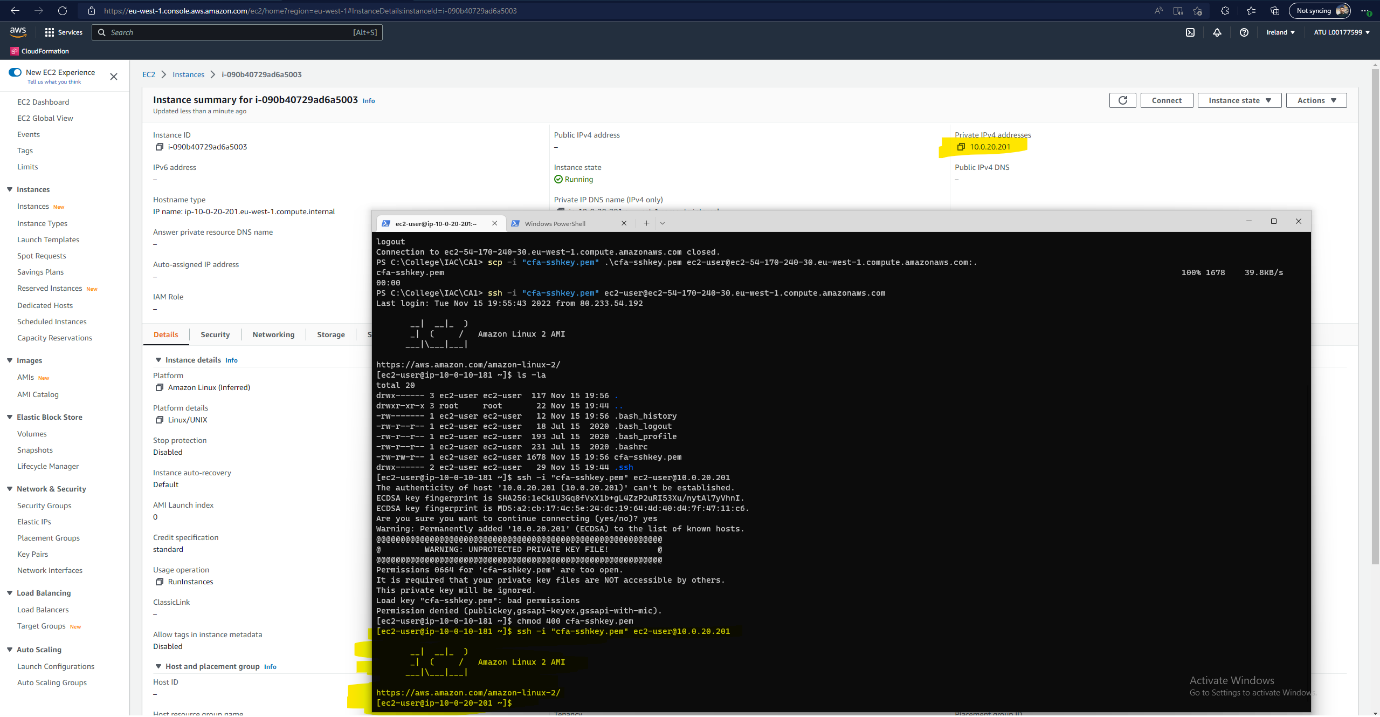


Figure SSH into Backend

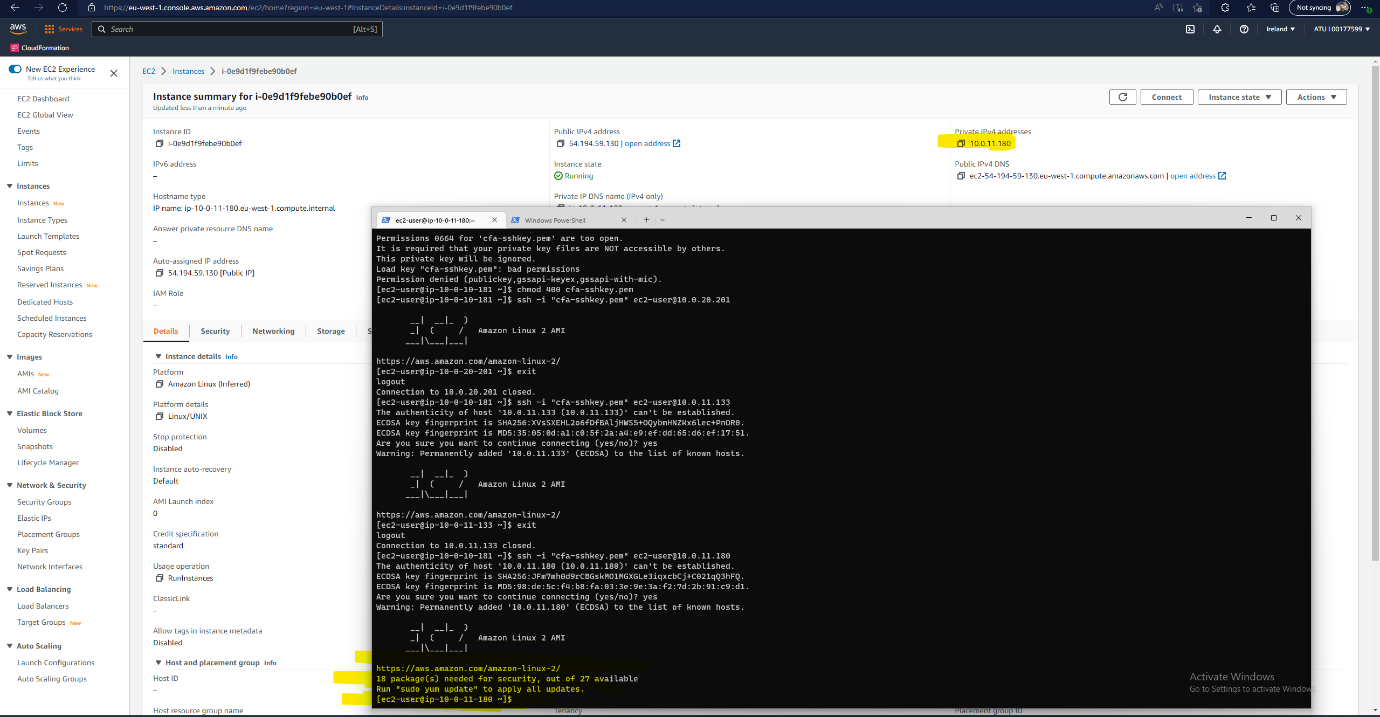


Figure SSH into Frontend

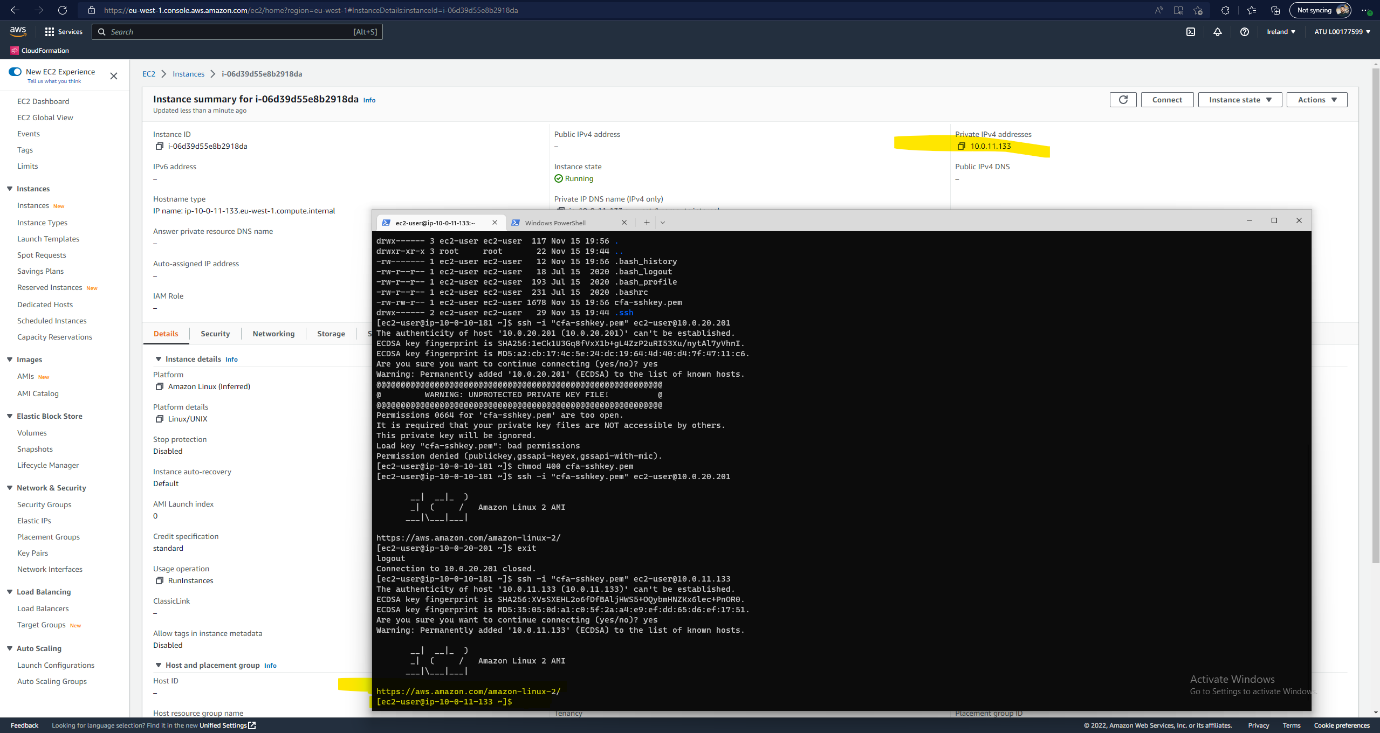


Figure SSH into Database Server

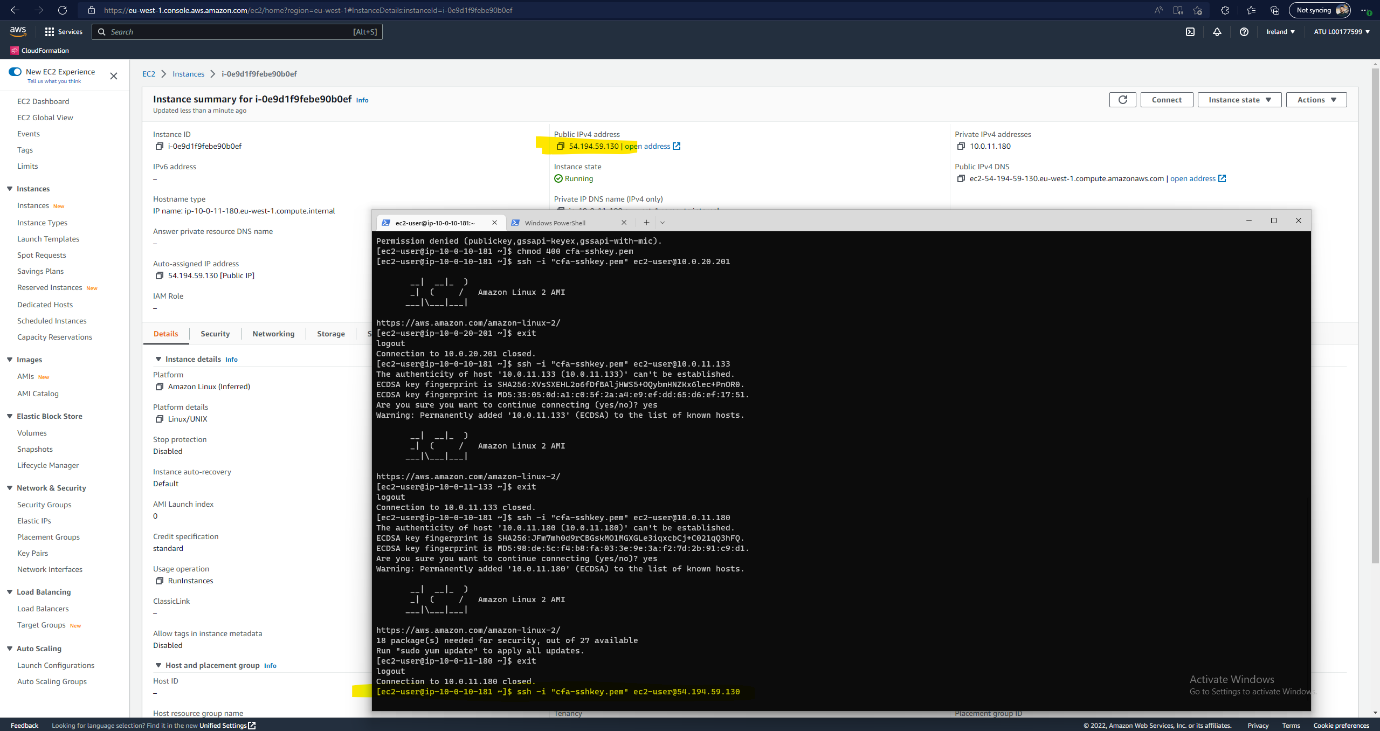


Figure Attempted SSH onto Public IP Frontend

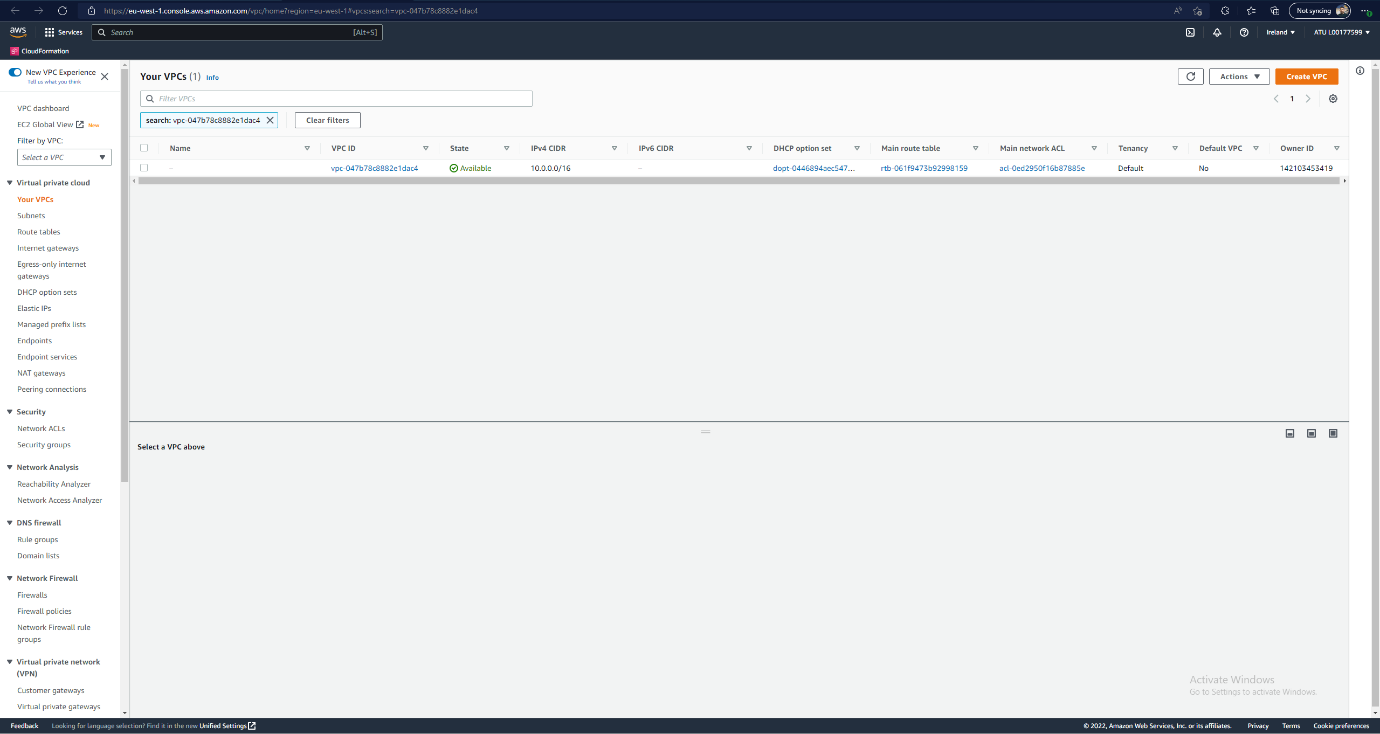


Figure VPC Console

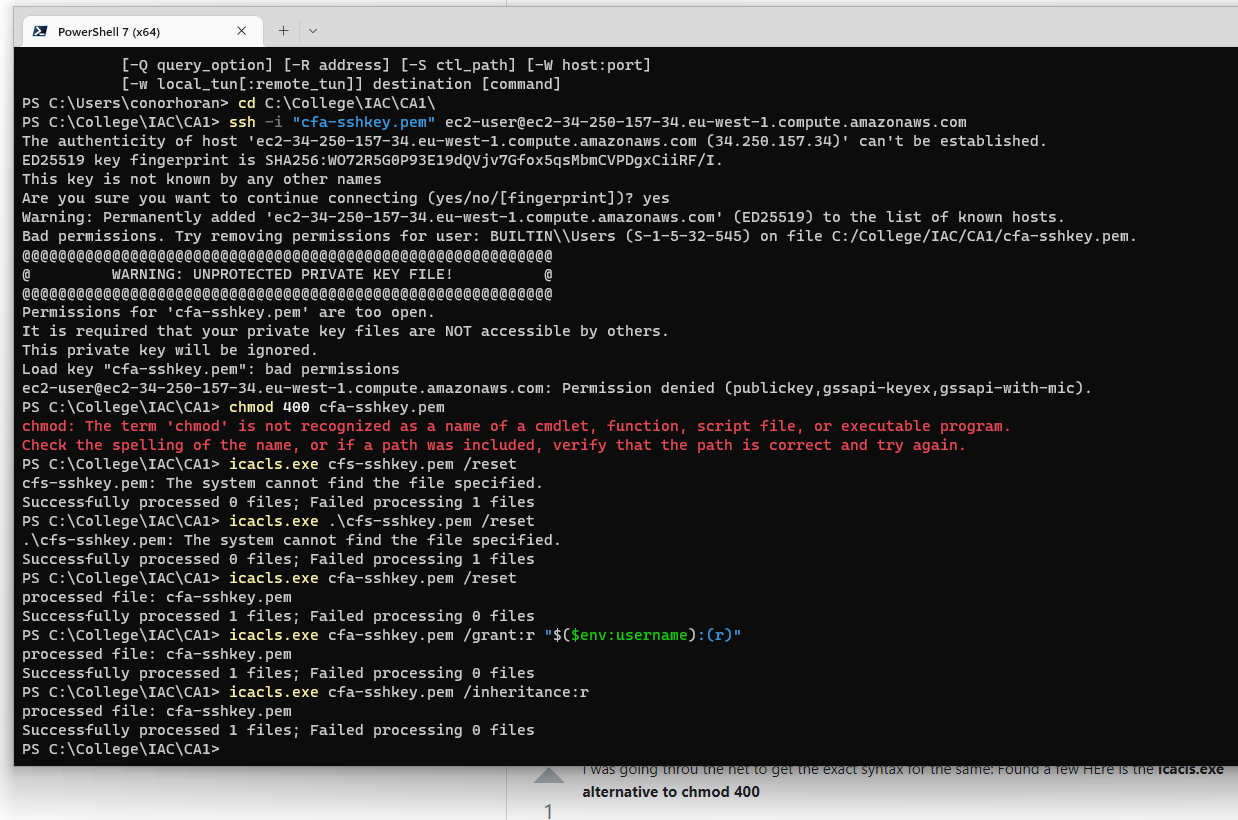


Figure Fix SSH Permissions Icacls.exe