# Cloud Security - CA1

Simon Lowry
MSCCYBE_JANOL_O
x21168938@student.ncirl.ie
National College of Ireland

## I. Approach & Project Planning

The security operations incorporated into manifesting this Cloud SaaS system ought to be workable and doable while meeting the requirements for modern software development. This includes adapting it into Agile working methodologies, which according to a recent study, 95% of organisations claim to be operating. There are organisations that are willing to have code make it's way to production systems even with security vulnerabilities, with 78% of organisations doing this according to a survey by Veracode & 48% acting this out ona regular basis, while, still somehow, believing their applications are secure. [1] [2]

Applying security needs to be doable and workable in parallel to the design of features and incorporating that very design while still being capable of being done at some degree of speed and flexibility. From this deploying relatively lightweight activities such as threat modeling can be implemented with relative ease to meet evolving and ever changing requirements while still making sure security risks are adhered to and not forgotten. Operating with a shift left mindset will look to incorporate security from beginning to end of the development and deployment lifecycle. In doing so we can look to capture some of the security vulnerabilities early in the development lifecycle and thus save in both cost and complexity of attempting to fix or redesign features later. This can be a big bonus for detecting these vulnerabilities as 60% of defects tend to occur in the design phase of development. [3]

It's going to include a shared responsibility model whereby some of the security will fall directly on us within the organisation and then some of the other layers of security will be applied by the cloud provider, AWS. AWS is a cloud service provider of high renown and has a whole plethora of security tools, features and benefits which make it an ideal choice for the deployment of the SaaS application here. AWS provides more compliance certifications and security standards than other offerings including PCI-DSS, GDPR, NIST800-171, FIPS, SOC2, HIPPA/HITECH, ensuring our SaaS system would meet regulatory requirements across the globe. [4]

**Benefits of using AWS Cloud computing:**

Going from fixed expense to variable expense, changing from Capital expenditure (CapEx) to Operational expenditure (OpEx). We also obtain massive economies of scale through the aggregation of massive amounts of customers on AWS translating into lower pay as you go scales for customers. Capacity and availability issues can be elastically adjusted based on requirements at a given point in time. This gives us the freedom to expand or contract capacity as the need arises, reducing idle capacity and helping to maintain optimum efficiency. The increase in speed and agility given by AWS cloud computing makes for a setup where security needs (which can evolve over time) can be met as they change. As a result it can help to meet security weaknesses and gaps quicker. It also has a vast amount of options for meeting the confidentiality, integrity and availability needs that make up the CIA triad of security. It's very easy and fast to configure high availability with instances deployed in regions across the globe and CDNs as well and also backup and recovery options to meet our availability needs for example. It provides a lot of visibility, auditability and controllability through tools like CloudWatch as well as automation to meet our security and compliance requirements with tools like CloudInspector. It offers CloudEndure Disaster Recovery. There is a high level of control over where our data resides with global availability zones as well.

**Threat model for Cloud SaaS Development:**

Some potential issues:

- **Cryptographic failures:** using weak or poor algorithms with known vulnerabilities, not securing sensitive data in transit or at rest. Poor management of keys, secrets and credentials resulting in credential theft of stolen sensitive data or man in the middle attacks.
- **Using components with known vulnerabilities:** offers an easy and low hanging branch for attackers to exploit our system and allows even lower level attackers to be able to attack our system
- **Misconfiguration of networks and change control:** poorly configured networks. subnets or access

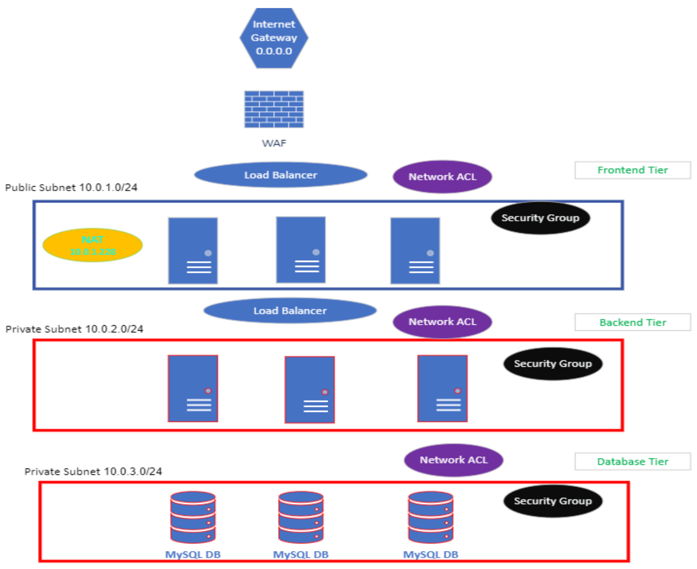
control lists can server as an easy win for attackers to exploit their cloud systems. Inadequate security around change control can also prevent non-repudiation and threats like insider threats can go unnoticed.

- **Inadequate logging and monitoring:** inadequate logging and monitoring can lead to nefarious activities and security breaches occurring without being detected or stopped.
- **Lacking secure software practices:** it's important that the SaaS application itself has security practices throughout it's lifecycle to reduce the likelihood of security vulnerabilities and weaknesses. These could result in XSS vulnerabilities, SQL injection vulnerabilities, session hijacking, cross site request forgery to name a few application specific threats.
- **Poorly configured identity and access management:** badly setup access management can lead to individuals receiving too much privileges and if an account is compromised it can then become a greater liability to the organisation.
- **Business continuity and resiliency:** ensuring that business continuity and disaster recovery practices are in place are paramount to the resiliency of the system. This includes being able to protect and defend against DDoS attacks as well.
- **Service and data integration:** using services in a poorly managed way which ends up compromising the security of the system or it's data.
- **Lack of cloud security architecture:** Setting up a secure cloud architecture from the get-go ensures that security is embedded in the setup of the systems being deployed into the cloud environment. This helps to try and maximise security and employ security principles, methods and practices in our cloud environment.

[5]

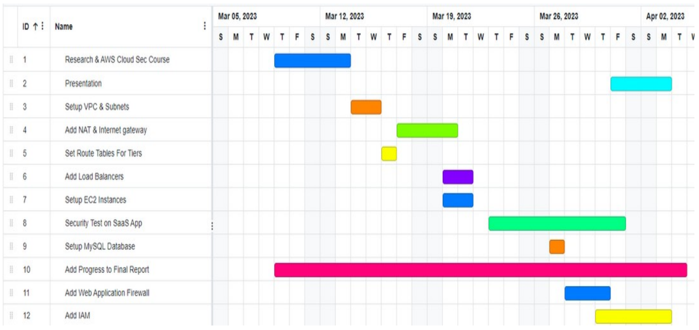
Mitigations for these threats are outlined in section 2.

## Cloud SaaS Architecture



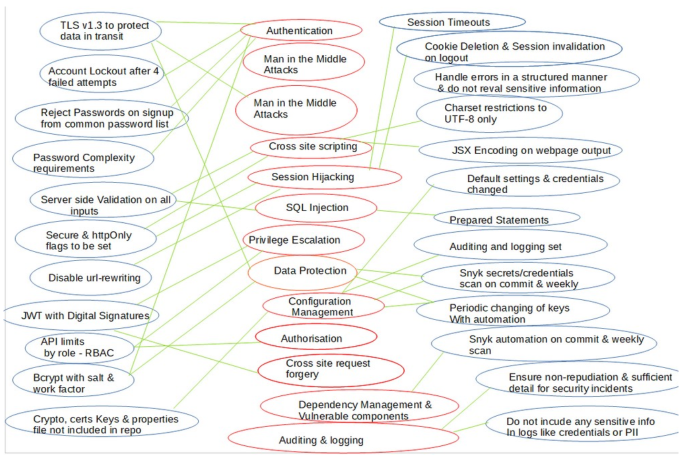
The architecture itself is setup to be a three tier architecture with the presentation layer housing the UI code and components, the next layer making up the business logic and APIs and then finally the database layer behind that. This establishes a separation of concerns with distinct sections for each part of the archiecture. Each tier can have it's own set of security concerns and these can be addressesed per their needs and evolved separately decoupled from the other layers. Both the presentation layer and business logic layer will be made to be auto-scalable. This can help in managing heavy loads of traffic were they to arise. It's automated so no manual interaction is required and can help to sustain availability for the disparate loads that these tiers could potentially face over time. By having these layers separate, they can expand and contract independently according to the evolving demands over time. More on the architecture and is expanded upon in section 2 with the choice of how it's been constructed elaborated on.

## Gantt Chart - Project Plan

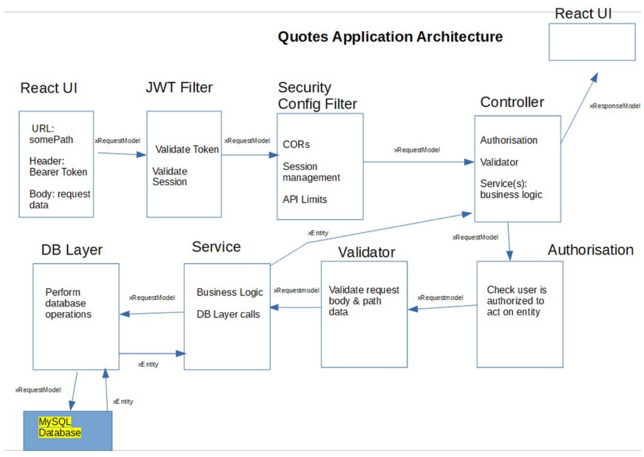


| ID ↑ | Name |
|---|---|
| 1 | Research & AWS Cloud Sec Course |
| 2 | Presentation |
| 3 | Setup VPC & Subnets |
| 4 | Add NAT & Internet gateway |
| 5 | Set Route Tables For Tiers |
| 6 | Add Load Balancers |
| 7 | Setup EC2 Instances |
| 8 | Security Test on SaaS App |
| 9 | Setup MySQL Database |
| 10 | Add Progress to Final Report |
| 11 | Add Web Application Firewall |
| 12 | Add IAM |

**SaaS Application Threat Model with mediation steps taken**


TLS v1.3 to protect data in transit
Account Lockout after 4 failed attempts
Reject Passwords on signup from common password list
Password Complexity requirements
Server side Validation on all inputs
Secure & httpOnly flags to be set
Disable url-rewriting
JWT with Digital Signatures
API limits by role - RBAC
Bcrypt with salt & work factor
Crypto, certs Keys & properties file not included in repo
Authentication
Man in the Middle Attacks
Man in the Middle Attacks
Cross site scripting
Session Hijacking
SQL Injection
Privilege Escalation
Data Protection
Configuration Management
Authorisation
Cross site request forgery
Dependency Management & Vulnerable components
Auditing & logging
Session Timeouts
Cookie Deletion & Session invalidation on logout
Handle errors in a structured manner & do not reval sensitive information
Charset restrictions to UTF-8 only
JSX Encoding on webpage output
Default settings & credentials changed
Prepared Statements
Auditing and logging set
Snyk secrets/credentials scan on commit & weekly
Periodic changing of keys With automation
Snyk automation on commit & weekly scan
Ensure non-repudiation & sufficient detail for security incidents
Do not incude any sensitive info In logs like credentials or PII

**Architecture of the SaaS Application - Quotes**



To give a brief summary of the SaaS application Quotes: it has its own secure architecture, it's making use of spring security and leveraging this tailored framework that's specifically built with security as it's priority. It has a JWT token filter, session management, api limits with deny by default, CORS is in place as well, validation occurs for all untrusted data and authorization is also checked before it enters the business logic layer. Protections against a multitude of attacks are in place as well with secure design principles applied. React also provides encoding for all UI outputs as well for more protection and there is routing on the frontend and checks for the presence of a token for making requests to mention a few elements of it's security in place.

## II. Ease Selections of Tools, Methods, Frameworks & Benchmarking

Instilling secure design principles is of paramount importance to the secure SaaS system in a cloud environment. Some of the design principles that would applied to this SaaS system would be least privilege, minimising the attack surface, defence in depth, meeting the triple A of security in accounting, authentication and authorization, applying adequate cryptographic methods to protect the data in transit and at rest, properly preparing for security events as well. Authorisation will be found in the use of IAM and applying role based access control (RBAC) throughout. By applying role based security it ensures that the maintenance of privileges is carried out on a group basis, helping to protect against unnecessary privileges afforded to different individuals. It also helps with the maintenance of privileges across large groups of people. Here least privilege would also be applied to ensure that the roles only have the necessary privileges to perform their duties and no more.

Defence in depth will be applied in a multitude of ways, giving the system a number of layers of security to protect the system. An example of this would be the web application firewall which will be able to protect against a variety of application based attacks such as sql injection and cross site scripting. On top of that there will also be load balancers behind that which will help distribute the web traffic proportionally and protect against DDoS attacks. Behind that network segmentation will again provide further protection and so on.

Minimising the attack surface will help to reduce the spectrum of possibilities for attackers and reducing the likelihood of unearthed vulnerabilities being exploited. AWS operates with a deny by default setup on it's interactions with network components. Applying only the necessary communications to the EC2 instances which will house the parts of the applications, the subnets, load balancers to only the minimum traffic will again enhance our security. It also makes the attackers life much more difficult with only a limited amount of ports, protocols and IPs accepted at the various layers of the architecture.

Applying industry standard cryptographic methods, algorithms and practices will be a core aspect of protecting the data in transit and at rest. Earlier versions of SSL had some core vulnerabilities like the Heartbleed bug highlighting the importance of using the most up to date version of TLS. That's currently TLS v1.3.

Beyond that, there will also be the use of official certificate authorities and public key infrastructure in place to enhance the overall security of the application. Data will be protected with industry standard algorithms that can help to encrypt the data at rest as well. AWS Key management system will keep the keys protected as well and AWS Secrets manager can easily generate them. Proper authentication will be required to access the keys and this will be limited to admins.

Effective logging both in the SaaS application itself and throughout the VPC and the various components helps with identifying security incidents. Whether it be applying logging and monitoring in the database for prolonged CPU usage or a lack of memory available can lead to application availability issues. Setting up this logging to be combined with subscriber based email notifications can help with monitoring these situations as they arise. The same applies with actions throughout the VPC which ought to be tracked and auditable and this will be part of our security tracking in the architecture.

The SaaS application has also applied security throughout its development with abuse cases, misuse cases, threat modelling being incorporated in its design of features. To complement that there is also the use of frameworks tailored specifically for security like Spring security, and using a UI language and framework that caters for security features by default like ReactJS.

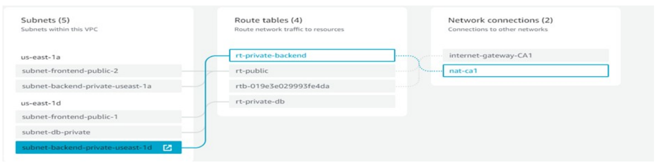
A **VPC or Virtual Private Cloud** gives us an isolated section in AWS Cloud where we can orchestrate our virtual networks which we are defining. In our VPC we're able to add network gateways, route tables, subnets and IP addresses ranges. We're able to customise our own network configurations and apply defence in depth with a multitude of technologies to increase the overall security of our SaaS application being deployed to the cloud. We can also create subnets in different availability zones for high availability and greater response times.

An internet gateway provides two different purposes. Firstly, it gives an entry and exit point for internet routable traffic within your VPC route tables. Secondly, it also performs network address translation for instances that have a public IP. This can help systems with IP addresses from being directly targeted for traffic or attacks from the internet. For instance, in our design, the database will not be directly accessible to the wider internet.

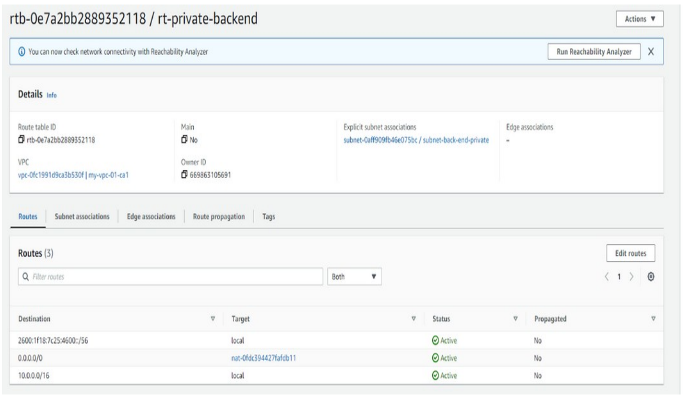
The gateway is given an elastic IP address which is a static IP address that does not change over time. This is important for the availability of an application so that it's accessible via the same entry point and IP.

A **route table** is made up of a set of rules also known as routes and can be used to delimit the how network traffic will be allowed to move throughout the subnets and VPC and is also used here to delineate the direction traffic will be allowed to flow. This can play an important role in limiting the traffic. Careful setup was needed here to only set traffic in the correct directions and no more.

**Security groups** act as a virtual firewall which controls inbound and outbound traffic. These perform their actions at the instance level as opposed to the subnet level. Each instance can therefore be applied to have their own virtual firewall protecting them. This helps to further delimit the allowed traffic to your instances. Security groups can allow based on IP, IP ranges and ports and protocols as well. All rules are evaluated before the decision is made to allow the traffic.A misconfiguration at this point could have dangerous consequences. This can help protect against malicious attacks when we apply least privilege. Here we can only allow the necessary IPs, ports and protocols to be able to operate.



The resource map for our vpc now shows that our backend subnet is associated with the nat-ca1 and will look to route it's traffic through there as shown below in it's route table.


rtb-0e7a2bb2889352118 / rt-private-backend

You can now check network connectivity with Reachability Analyzer

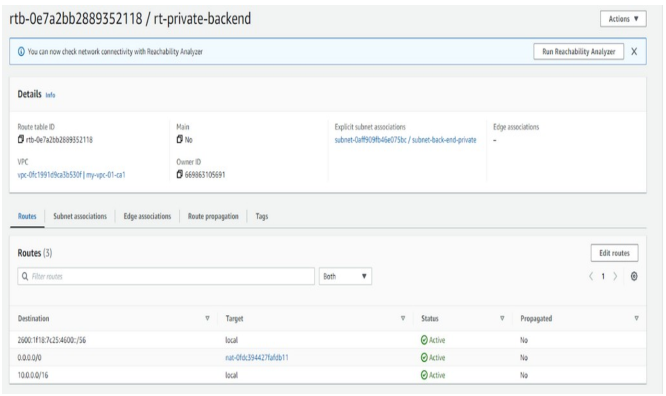
Details

Route table ID: rtb-0e7a2bb2889352118
Main: No
Explicit subnet associations: subnet-0aff909fb46e075bc / subnet-back-end-private
Edge associations: -
VPC: vpc-0fc1991d9ca3b530f | my-vpc-01-ca1
Owner ID: 669863105691

Routes (3)

| Destination | Target | Status | Propagated |
|---|---|---|---|
| 2600:1f18:7c25:4600::/56 | local | Active | No |
| 0.0.0.0/0 | nat-0fdc394427fafdb11 | Active | No |
| 10.0.0.0/16 | local | Active | No |

Public subnet which will contain the front end code, will have it's traffic going to the internet gateway and onwards to the public internet.



rtb-0e7a2bb2889352118 / rt-private-backend

You can now check network connectivity with Reachability Analyzer

Details

Route table ID: rtb-0e7a2bb2889352118
Main: No
Explicit subnet associations: subnet-0aff909fb46e075bc / subnet-back-end-private
Edge associations: -
VPC: vpc-0fc1991d9ca3b530f | my-vpc-01-ca1
Owner ID: 669863105691

Routes (3)

| Destination | Target | Status | Propagated |
|---|---|---|---|
| 2600:1f18:7c25:4600::/56 | local | Active | No |
| 0.0.0.0/0 | nat-0fdc394427fafdb11 | Active | No |
| 10.0.0.0/16 | local | Active | No |

**SaaS application security considerations**

Making use of an industry tested framework like Spring Security which has been battle hardened and specifically designed for security helps within our SaaS application. It's also employing ReactJS which has security features embeeded in it and MySQL as the database selection is also the same.

The **EC2** instances of the frontend/presentation tier and the backend tier are both set to be parts of Auto Scaling groups. They are both using Amazon Linux 2023 AMI which is the latest version of that OS and thus up to date with the most recent security patches, it's also the free tier for the purposes of this project. It's access has been set to RSA keypair and OpenSsh. [A1] Detailed cloud watch monitoring has also been selected to increase the logging detail that could potentially be used for detecting and resolving security issues. In the "User data" section, a bootstrap script was added as well. This bootstrap script prompted the installation, management, and configuration of tools useful for cluster monitoring and data loading, further increasing logging and monitoring capabilities. [A3] Since this is a prototype setup for a college project the number of instances in the auto scaling group is set to have a desired and minimum capacity of 1, but it's also capable of expanding that to a total 5, automatically, should that be required. This enhances the availability of our application at each layer. [A4]

In terms of **IAM**, we'll be operating by RBAC as mentioned and this would involve IAM Groups for which there will be three groups created: Admins, Developers & Testers. Least privilege will be applied such that they are only given the necessary permissions to perform their duties and no more. Admins would have the ability to manage instances, infrastructure and modify and create them. They can also add, or delete users from the various groups in the account. Developers would be given permissions to work on and operate on a dev version of the application and push to that environment. They would also be able to push code to the github repo. Testers would be another group and would have permission to perform QA testing when the application has reached the Test Server. Neither developers nor Testers would have any access to anything at the infrastructure level. Promotion of code from dev to test to production would require signing off by an admin would the privileges to do so. Developers would be able to read and write to databases at both the dev and test level but not the production level. They would not be able to modify the schemas of the database. Testers would have no access to any of database. MFA would be required for all account authentication and then in particular, for more sensitive APIs and services, it would need to be re-authenticated as well.

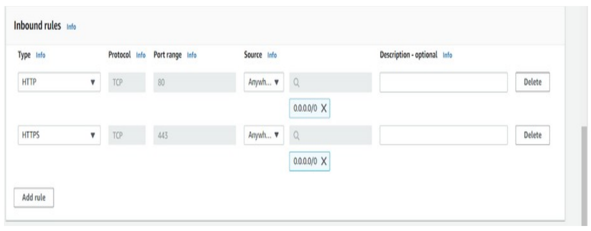
There's also the use of a **Web application firewall** in the VPC. This is set to protect the application web application attacks such as Cross Site Scripting and SQL injection in web traffic. There's also bot mitigation capabilities to help protect against pervasive bots which can be monitored and blocked where required and this includes for crawlers, scanners and web scrapers as well. [6]

**Network segmentation** divides the VPC into different subnets. Isolating each tier of the architecture ensures that we can apply security requirements that are specific to each individual tier in isolation. On top of that, if an attacker were to compromise one network, they would only be able to potentially interact with systems on that network, it provides isolation from this point of view and makes lateral movement more difficult for attackers. Some of the subnets are designated as public subnets and others as private subnets. The public subnets are directly connected to the internet gateway and thus the wider internet and can be directly interacted with. The private subnets are across two layers, one layer for the backend servers housing the various APIs and business logic of the application. The other layer contains the databases used for the application. These private subnets are hidden behind the NAT gateway that's been deployed and as such can not be directly interfaced with and any requests for them must come through the NAT gateway, load balancer, network ACL and security groups. These are all providing defence in depth as means of protecting these inner layers of the infrastructure.

**Load Balancers** play an important role in protecting the smooth running of our applications. They ensure the load of traffic is distributed amongst the various instances of a given application to ensure that none end up getting overwhelmed in the process. This can help against DDoS attacks where attackers look to bombard a system with requests from a whole host of devices (usually a botnet) to take the system offline. This can result in compromising the availability of our application and AWS application load balancers help here. The load balancers applied for the backend subnets also act as a gateway to the subnets and have security groups applied to them ensuring that requests can only come from

NAT and all other traffic is rejected. The EC2 instances cannot be directly communicated with from services on the internet instead they must be funnelled in through our networks in a fashion that's been designed to minimise the attack surface to our cloud SaaS application.

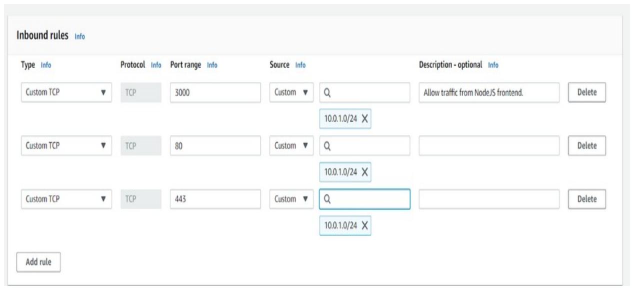
This first load balancer is internet facing since this is going to be on the presentation tier layer, between the gateway and the internet. The other one is placed between the frontend tier and the backend tier. That one is set to an internal scheme. Both are Application load balancers.



**Inbound rules**

| Type | Protocol | Port range | Source | | Description - optional | |
|---|---|---|---|---|---|---|
| HTTP | TCP | 80 | Anywh... | 0.0.0.0/0 | | Delete |
| HTTPS | TCP | 443 | Anywh... | 0.0.0.0/0 | | Delete |

HTTP on port 80 and HTTPs on port 443 is set to allow for incoming traffic to the load balancer associated with the front end subnet.



**Inbound rules**

| Type | Protocol | Port range | Source | | Description - optional | |
|---|---|---|---|---|---|---|
| Custom TCP | TCP | 3000 | Custom | 10.0.1.0/24 | Allow traffic from NodeJS frontend. | Delete |
| Custom TCP | TCP | 80 | Custom | 10.0.1.0/24 | | Delete |
| Custom TCP | TCP | 443 | Custom | 10.0.1.0/24 | | Delete |

The backend load balancer will be allowing traffic from frontend server node js to the load balancer on the backend. The above images are both from the security groups setup for these individual load balancers.

**Database security and selection**

One of the core aspects of any SaaS deployment is the database. Selecting an appropriate database and configuring it correctly is vital to the security of a given system. By leveraging the capabilities on offer help to ensure that the database is properly protected can help with maintaining the confidentiality, integrity and availability of the data contained within the database as well as keeping the database itself secure.

When considering some of the primary NoSQL options, security has appeared to be an afterthought for some NoSQL databases like MongoDb and not a priority. This popular NoSQL option MongoDB for example has endured a whole host of attacks over 2014 to 2017 and as a result of that only then retrospectively added in. This lack of security amounted to almost 30,000 MongoDBs being hacked and also experiencing ransomware attacks. This included an almost ludicrous figure of 22800 production systems of MongoDb which were live, having no credentials at all. While they have improved some of the their capabilities since then, a system with this level of being dismissive of security is not a good option to go with.

MySQL on the other hand offers a full suite of functionality that's been central to their design and features from the get go. It's also a plus when it comes to integrity and greater data reliability. It gives you TLS support, firewall rules, auditing capabilities of a high level for tracking operations and internal functioning. This can monitor tampering with the database schema, attempts to access a given table, or sign out attempts and authentication efforts as well. All of which can help identify nefarious activity of attackers or insider threats alike. There are also a number of authentication options, backup options for both online and offline, privilege restrictions, account locking, resource limits on accounts. All of these make MySQL a strong option for the database selection here, especially in regard to security concerns. This is why it was selected for this SaaS application.

Next, we'll dive into how it's been configured. System hardening has been applied to try and maximize security. One availability consideration, is the size of the database and if that limit were to be exceeded, how would the system handle it? In order to sustain availability of this primary component with the database, storage autoscaling is enabled. This will increase the storage beyond the 200 GB allocated storage where needed automatically. For the storing of admin credentials within the database, AWS Secrets manager is being used. This offers a centralized means of managing the lifecycle of secrets. They are securely encrypted (with a KMS key), rotated regularly and replicated to support disaster recovery and multi-region applications. [7]

Securing the data as it's in transit is an important consideration to protect against Man in the Middle attacks and potential tampering or misuse of data being snooped. Here we're looking to setup TLS to encrypt the data as it moves between the database instance and the application layer. The certificate used is obtained from an official certificate authority and contains a private key algorithm of RSA 4096 with SHA384 signing algorithm. This provides strong encryption and a powerful integrity faculty with SHA384. The CA used also supports automatic rotation of certificates ensuring that the identity of the database is persisted and the communications are continually secure as well as the data at rest.

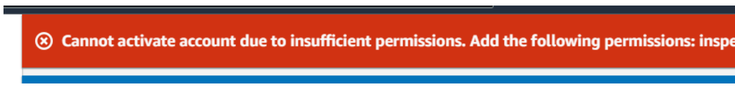
The database credentials require password authentication and also making use of IAM users and roles to manage the authentication and authorization to the database resources. Enhanced monitoring has been selected to increase the visibility on CPU usage. Automated backups are also in use to protect against any disaster recovery scenarios and they are retained for 7 days. Log exports for CloudWatch are set for auditing, errors, general logging and slow query loggins. This can help to setup alerts and monitor potential security issues and respond swiftly when it occurs. A low hanging branch for attackers is exploiting out of date software with known vulnerabilities. This database has auto minor upgrade enabled to ensure new minor versions are upgraded to as they are released providing protection against known exploits over time. Another additional protection added is to ensure there is deletion protection. Protect from insider threats and inadvertent deletions. loss of availability, integrity of data. Some CloudWatch alarms have been setup to monitor and notify for high cpu utliization and also for the database running out of memory. This can protect against potential availability issues and there is an email being sent to my email address currently, this would be expanded out to a full team of people's emails in a production system. [A7]

## III. Technical Testing Approach

The SaaS application itself has been scrutinized with SAST and DAST tools as well as a Penetration Testing Checklist obtained from OWASP. Snyk is a powerful SAST tool which provides secrets and credentials scans, as well as analysis of code for vulnerabilities and also the capability to assess infrastructure such as containers & it also checks for known vulnerabilities in dependencies in the project. The DAST applied in this case was ZAP which can give immediate feedback of some potential security vulnerabilities in the application while running. It has a plethora of payloads which can be applied to an application to see if it has a whole host of different categories of security vulnerabilities. These tools have been applied to assess the security of the SaaS application and Snyk is configured to run periodically. The next phase for this would be to integrate these tools into a github book which could trigger a jenkins pipeline build on commit. That would result in the application being built up and containerized each push of a commit and then followed up with multiple security tools being run on each commit of code. This continuous integration approach with security included allows for security vulnerabilities as soon as possible by these tools and gives the chance of developers to fix these vulnerabilities immediately when identified.

Amazon inspector is an automated security assessment tool for EC2 instances and applications. It can help improve the security and compliance of applications deployed on AWS. It can help to identify security vulnerabilities and deviations from the various security best practices with your applications. This for both when they are deployed and when they are also in a production environment. An example of how it can help would be for unintended network accessibility of your EC2 instances and for vulnerability on those instances. It helps to proactively address security issues before they are able to properly impact on your application.

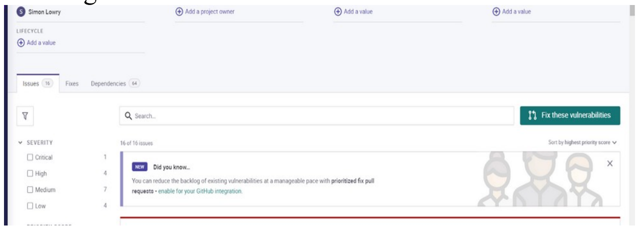
After it performs an assessment it will give a set of security issues and also the risk level associated with each vulnerability. These can be compiled directly or as part of assessment reports through the AWS console or API. It can also be deployed within dev ops pipelines to help iteratively assess your applications as part of the deployment process . [9] [8] There is a 15 day free trial, however, again on this lab account, it's not possible to run this. It requires a specific permission which can't be added with this lab account:


Cannot activate account due to insufficient permissions. Add the following permissions: inspe

There was a hope to be able to apply Nessus vulnerability scanner instead. This could help to identify vulnerabilities in the setup and infrastructure. However, in order to setup Nessus within AWS, it requires to be able to add an IAM role which is not possible to do so with this lab based account used for this project. [10]

## IV. Findings and Risk Rating

The automated tool Snyk was the means of performing static application security testing. Snyk has the capacity to find and fix security vulnerabilities for our SaaS system on an ongoing basis. It can increase the capabilities of security reviews and provides a means finding security vulnerabilities earlier in the lifecycle and thus reducing the cost. Snyk performs it's operations on code, containers and configurations and makes use of the snyk vulnerability database to help find open source vulnerabilities. The github repo has been integrated with this tool and set to run weekly automatically. It also has the capacity to run on each commit and push if a pipeline were triggered with git hooks as well. It's able to find security issues within the code and dependency problems. When updating verisons of different dependecies, it would be best to test these locally first. This can help to find out whether any breakage of functionality occurs. When it comes to the code related security issues, address the ones which are deemed to be the highest and critical risk first and then proceed to the rest. Determine whether or not you're dealing with a false positive before proceeding to make any changes to the code.  When false positives are identified mark this off in the snyk UI and provide a reason as to why it is the case for future reference and tracking.


Simon Lowry | Add a project owner | Add a value | Add a value

LIFECYCLE
Add a value

Issues 16 | Fixes | Dependencies 64

Search... | Fix these vulnerabilities

SEVERITY
- Critical 1
- High 4
- Medium 7
- Low 4

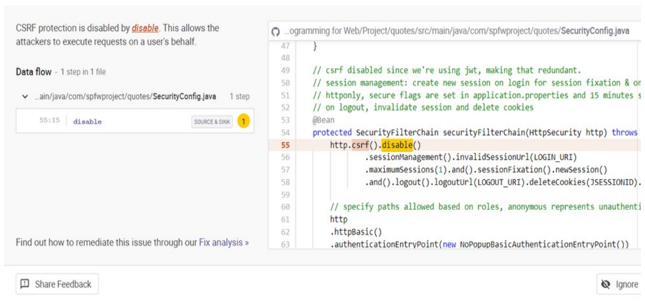
16 of 16 issues | Sort by highest priority score

NEW Did you know...
You can reduce the backlog of existing vulnerabilities at a manageable pace with **prioritized fix pull requests** - enable for your GitHub integration.

Snyk has identified 16 potential issues. 1 issue classified as Critical, 4 as High, 7 as medium and 4 low risk. On top of that, 64 potential dependency issues. The issues were addressed by priority with the critical issues assessed first, then high risk and so on.

The use of the package org.springframework:spring-webmvc could lead to allowing unauthroirzed access in that version. There is a total or partial loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact. For example, an attacker steals the administrator's password, or private encryption keys of a web server.  [A7] This package is enclosed within spring boot web package. Spring boot web package has been subsequently updated to combat this vulnerability.

Beyond that there was also some high vulnerabilities, some of which were again related to dependencies and referenced improper input validation, dos and authorization bypass. These dependencies have been updated.


CSRF protection is disabled by *disable*. This allows the attackers to execute requests on a user's behalf.

Data flow - 1 step in 1 file

...ain/java/com/spfwproject/quotes/SecurityConfig.java 1 step

55:15 disable SOURCE & SINK 1

Find out how to remediate this issue through our Fix analysis »

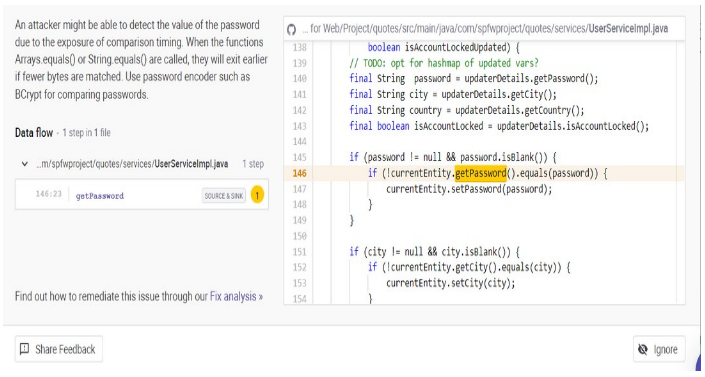
...ogramming for Web/Project/quotes/src/main/java/com/spfwproject/quotes/SecurityConfig.java

```
    }

    // csrf disabled since we're using jwt, making that redundant.
    // session management: create new session on login for session fixation & on
    // httponly, secure flags are set in application.properties and 15 minutes s
    // on logout, invalidate session and delete cookies
    @Bean
    protected SecurityFilterChain securityFilterChain(HttpSecurity http) throws
        http.csrf().disable()
                .sessionManagement().invalidSessionUrl(LOGIN_URI)
                .maximumSessions(1).and().sessionFixation().newSession()
                .and().logout().logoutUrl(LOGOUT_URI).deleteCookies(JSESSIONID).

        // specify paths allowed based on roles, anonymous represents unauthenti
        http
        .httpBasic()
        .authenticationEntryPoint(new NoPopupBasicAuthenticationEntryPoint())
```

Share Feedback | Ignore

Another issue previously identified was for potential cross site request forgery. the tool raises a concern about having set csrf protections in spring security to disabled. This is a false positive as the tool is not able to realise that JWT is being used instead. This serves as CSRF protection preventing users from being tricked into taking an action on the website when authenticated. Attackers would not be able to craft a link which contains the JWT tokens as they are only accepted via the Authorization headers of http requests.

Snyk also raised an issue about potential hardcoded passwords being contained in the code base. This was due to a setPassword setter method being called. However, these were in testing code where the passwords being set where mock/fake passwords for the sake of testing on performing login failures.


An attacker might be able to detect the value of the password due to the exposure of comparison timing. When the functions Arrays.equals() or String.equals() are called, they will exit earlier if fewer bytes are matched. Use password encoder such as BCrypt for comparing passwords.

Data flow - 1 step in 1 file

...m/spfwproject/quotes/services/UserServiceImpl.java 1 step

146:23 getPassword SOURCE & SINK 1

Find out how to remediate this issue through our Fix analysis »

... for Web/Project/quotes/src/main/java/com/spfwproject/quotes/services/UserServiceImpl.java

```
        boolean isAccountLockedUpdated) {
    // TODO: opt for hashmap of updated vars?
    final String  password = updaterDetails.getPassword();
    final String city = updaterDetails.getCity();
    final String country = updaterDetails.getCountry();
    final boolean isAccountLocked = updaterDetails.isAccountLocked();

    if (password != null && password.isBlank()) {
        if (!currentEntity.getPassword().equals(password)) {
            currentEntity.setPassword(password);
        }
    }

    if (city != null && city.isBlank()) {
        if (!currentEntity.getCity().equals(city)) {
            currentEntity.setCity(city);
        }
```

Share Feedback | Ignore

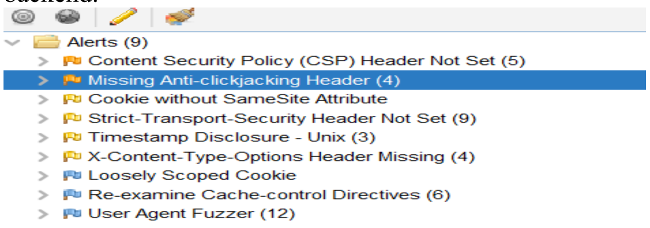
A passive timing based attack whereby attackers may be able to discern passwords based on .equals exiting earlier. Instead the tool suggests using the bcrypt encoder equals method which is what this will be changed to.

**Dynamic Application Security Testing**

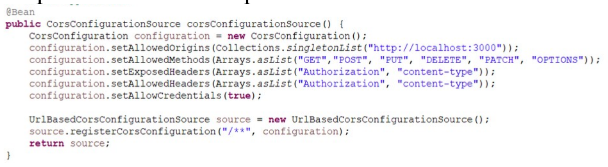
The dynamic application security testing was conducted with OWASP ZAP tool. [8]

Here's a screenshot of it run on /auth/login, the login API on the

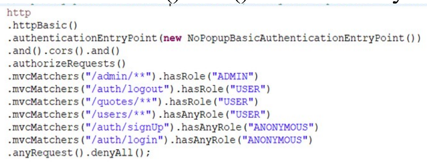
backend.


Alerts (9)
- Content Security Policy (CSP) Header Not Set (5)
- Missing Anti-clickjacking Header (4)
- Cookie without SameSite Attribute
- Strict-Transport-Security Header Not Set (9)
- Timestamp Disclosure - Unix (3)
- X-Content-Type-Options Header Missing (4)
- Loosely Scoped Cookie
- Re-examine Cache-control Directives (6)
- User Agent Fuzzer (12)

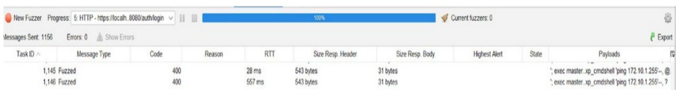
The active scan of my login API showed that I had not properly configured my CORS setup. While the bean with all the setup for CORS was in place:


```
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(Collections.singletonList("http://localhost:3000"));
    configuration.setAllowedMethods(Arrays.asList("GET","POST", "PUT", "DELETE", "PATCH", "OPTIONS"));
    configuration.setExposedHeaders(Arrays.asList("Authorization", "content-type"));
    configuration.setAllowedHeaders(Arrays.asList("Authorization", "content-type"));
    configuration.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}
```

I hadn't added .and().cors() to the securityFilterChain


```
http
.httpBasic()
.authenticationEntryPoint(new NoPopupBasicAuthenticationEntryPoint())
.and().cors().and()
.authorizeRequests()
.mvcMatchers("/admin/**").hasRole("ADMIN")
.mvcMatchers("/auth/logout").hasRole("USER")
.mvcMatchers("/quotes/**").hasRole("USER")
.mvcMatchers("/users/**").hasAnyRole("USER")
.mvcMatchers("/auth/signUp").hasAnyRole("ANONYMOUS")
.mvcMatchers("/auth/login").hasAnyRole("ANONYMOUS")
.anyRequest().denyAll();
```

1156 payloads of the SQL injection variety assaulted the application's login API and these were injecting their payloasd into the username and password parameters.[A6]



```
New Fuzzer  Progress: 5: HTTP - https://localh..8080/auth/login   100%   Current fuzzers: 0
Messages Sent: 1156   Errors: 0   Show Errors                                      Export
Task ID   Message Type   Code   Reason   RTT      Size Resp. Header   Size Resp. Body   Highest Alert   State   Payloads
  1,145   Fuzzed         400             28 ms    543 bytes           31 bytes                                  '; exec master..xp_cmdshell 'ping 172.10.1.255'--, @
  1,146   Fuzzed         400             557 ms   543 bytes           31 bytes                                  '; exec master..xp_cmdshell 'ping 172.10.1.255'--, ?
```

This did not even get one error as you can see above. This provides some level of confidence that the protections in place for securing against SQL are having an effect for the login page.

Then this was followed up XSS payloads:

This mounted a total of 1296 payloads through the same API and again established a measure off assurance that there is reasonable levels of security against these XSS attacks for this API.

## *V. Challenges and Limitations*

The Web application firewall is only operating with the basic functionality available to the free tier. This can be extended to get a whole suite of additional methods of blocking various kinds of potential threats from a variety of different providers which have services through AWS WAF which can increase the web application protection. It can also be extended to offer Rate-based Rules and Rate limiting capabilities. These can allow you to tailor specific rules which can block or add exemptions to blocking specified traffic based on IP addresses that reached certain thresholds. This can be used to further protect against DDoS attacks, malicious bots or brute force login attempts as well. The WAF can also be extended to use Account Takeover Prevention which can monitor traffic to the given applications login page and is capable of detecting unauthorized access to user accounts with compromised credentials. They are useful for protecting against stuffing attacks, brute force login attacks and more. [6]

MySQL databases can only be scaled vertically which provides a drawback on availability. Currently we're only set to have a single database in operation. Multitenancy ought to be added to increase the availability of the database. To complement this, database replication of live data with more database situated around the globe would be another extension of the database setup currently. High availability and data redundancy is a necessity for any production systems.

They can also be susceptible to race conditions, DDoS attacks and SQL injection, all of which put the onus on the developers and operators to provide other means of protection against these attacks. Increasing WAF utilisation can help to protect more against these beyond the already existing protections in the SaaS application itself.

Currently the availability zones are only set to two zones within the US. This would be an issue with latency and potential availability if there were to be massive regional outages in these areas. For a production system with an actual budget these availability zones ought to be extended throughout the globe to improve the availability and disaster recovery capabilities of our SaaS application and also the performance of the application as well.

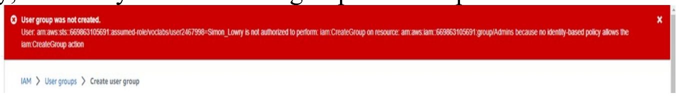
On top of that, there ought to be different environments for the different levels of the software process. It's important that chances that are made to the system are first fully setup and deployed to a test environment before reaching the production environment. Here any potential breakages of functionality, or security vulnerabilities can be both detected and addressed where needed before making their way to the production system. For major vulnerabilities that make their way to the production system and become immediately apparent, these ought to be rolled back swiftly followed by a thorough investigation into the root cause of the vulnerability.

Security tools ought to be introduced to the deployment pipeline. This can be done with pipelines such as Jenkins which can include tools such as SAST and DAST tools like Snyk and ZAP to try and identify vulnerabilities as good as pushed to the repo. This can take immediate assessment on a commit basis, performing a build in the process and giving the developer and the application security engineers immediate feedback on where their may be some vulnerabilities in the code or dependencies.

A network and application penetration test could help to identify other vulnerabilities that have thus far not been identified. Choosing a pentester with experience in testing in cloud environments could also be an advantage here. Beyond this, there could also be the use of both intrusion detection systems like snort or intrusion preventions like Tipping Point IPS to further increase the likelihood of detecting malicious activity. A SIEM could also be deployed and the alerts raised could be assessed by an incident response team. This incident response team would help with trained professionals that could identify false positives and properly investigate, escalate & remediate security incidents as they arise.

Security and compliance checklists and assessments like NIST or ISO could also be a way of identifying any other gaps that have not already been identified. A patch management program could be setup to ensure that roles, responsibilities and actions for patching is regularly addressed. VPNs could also be added for any direct access to a given EC2 instance.

Due to the nature of using an account which is lab based only, the ability to create IAM groups was not possible:



User group was not created.
User: arn:aws:sts::669863105691:assumed-role/voclabs/user2467998=Simon_Lowry is not authorized to perform: iam:CreateGroup on resource: arn:aws:iam::669863105691:group/Admins because no identity-based policy allows the iam:CreateGroup action

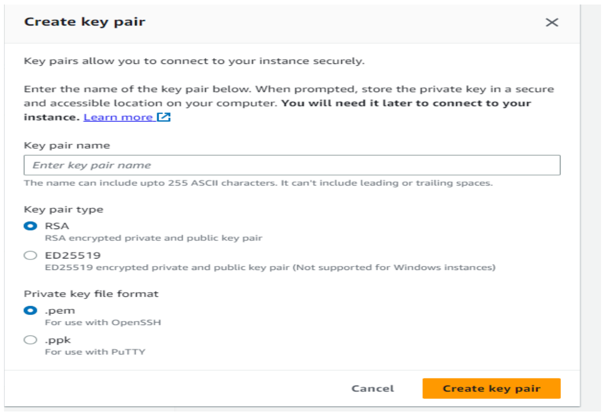
The only other way of including IAM would have been to set up an account with my own personal credit card & this was not going to be the case for a college project. The use of IAM would otherwise be applied for a real system being developed and deployed in a business setting. The permissions and groups were outlined in a previous section and would be fulfilled and expanded on in that scenario.

## VI. Conclusion and Findings

Setting up a SaaS application in a cloud environment is a complex task. The multitude of potential risks and concerns is huge and requires a lot of planning and coordination in order to firstly identify all of the risks, and then little by little, address them. The breadth of security controls required demands a vast amount of knowledge of a lot of tools, technologies and best practices specific to the cloud in order to be properly effective in securing the environment. This is also not a one time thing. With a lot of moving parts and evolving environments and systems, it requires ongoing and persistent vigilance. Something as simple as an overly liberal ACL or route table could be the cause of a major breach and the same applies to poorly managed credentials or configurations. The cost of getting these configurations wrong can result in major security breaches,revenue loss, reputational damage or GDPR fines for not adequately protecting customer data. Applying effective security practices and testing regularly is a must to decrease the risk of security breaches. On top of that, addressing those identified security vulnerabilities in a timely manner and not just sweeping them under the rug hoping an attacker won't find them. It's not a matter of if a security breach will occur but when.

## Appendix

### A1 - EC2 instance access setup to have RSA key pair access through OpenSsh



**Create key pair** ×

Key pairs allow you to connect to your instance securely.

Enter the name of the key pair below. When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** Learn more

Key pair name

*Enter key pair name*

The name can include upto 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

[✓ RSA] RSA encrypted private and public key pair

ED25519 ED25519 encrypted private and public key pair (Not supported for Windows instances)

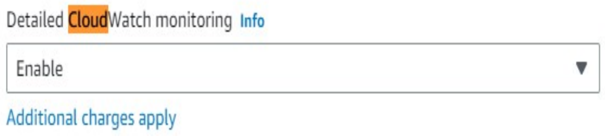
Private key file format

[✓ .pem] For use with OpenSSH

.ppk For use with PuTTY

Cancel | Create key pair

### A2 - Detailed cloud watch monitoring set to enabled for EC2 instances

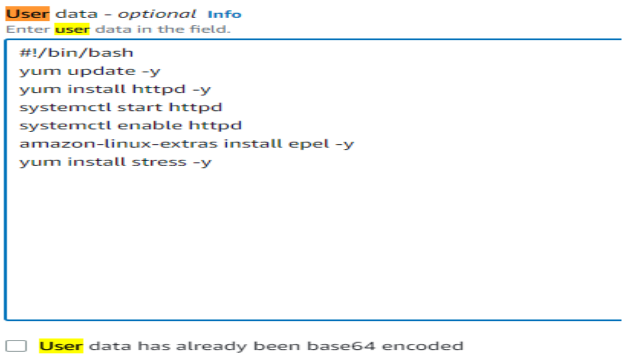


Detailed CloudWatch monitoring Info

Enable ▼

Additional charges apply

### A3 - EC2 instance "User data", the below bootstrap script was added to the user data. Bootstrap scripts allow installation, management, and configuration of tools useful for cluster monitoring and data loading.



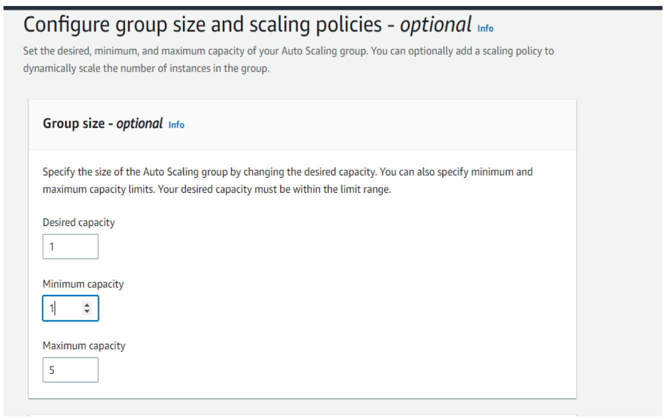
User data - *optional* Info

Enter user data in the field.

```
#!/bin/bash
yum update -y
yum install httpd -y
systemctl start httpd
systemctl enable httpd
amazon-linux-extras install epel -y
yum install stress -y
```

User data has already been base64 encoded

### A4 - EC instances auto scaling groups



Configure group size and scaling policies - *optional* Info

Set the desired, minimum, and maximum capacity of your Auto Scaling group. You can optionally add a scaling policy to dynamically scale the number of instances in the group.

**Group size - *optional*** Info

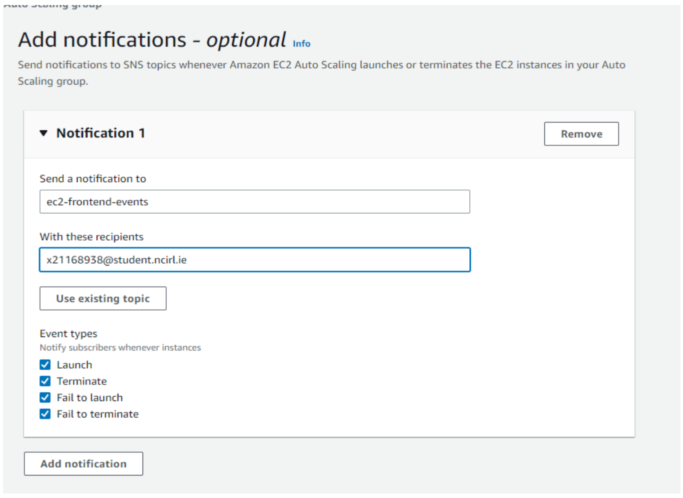
Specify the size of the Auto Scaling group by changing the desired capacity. You can also specify minimum and maximum capacity limits. Your desired capacity must be within the limit range.

Desired capacity: 1

Minimum capacity: 1

Maximum capacity: 5

### A5 - EC2 instances SNS topics for auto scaling



Add notifications - *optional* Info

Send notifications to SNS topics whenever Amazon EC2 Auto Scaling launches or terminates the EC2 instances in your Auto Scaling group.

**Notification 1** Remove

Send a notification to: ec2-frontend-events

With these recipients: x21168938@student.ncirl.ie

Use existing topic

Event types

Notify subscribers whenever instances

✓ Launch

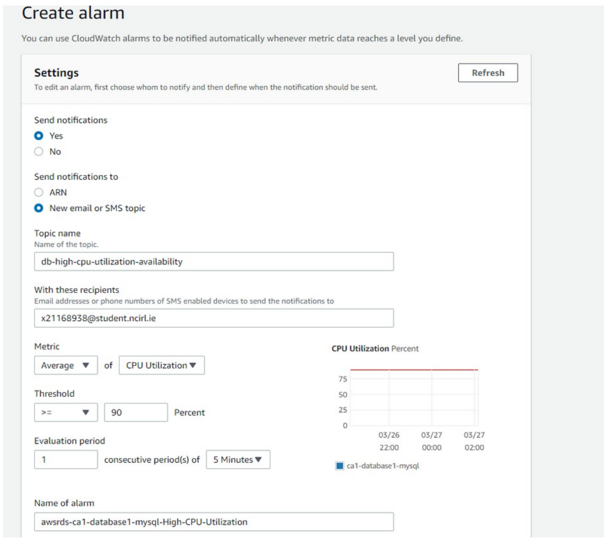
✓ Terminate

✓ Fail to launch

✓ Fail to terminate

Add notification

### A6 - Cloudwatch alarm based on CPU utilization in the database exceeding acceptable thresholds



Create alarm

You can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.

**Settings** Refresh

To edit an alarm, first choose whom to notify and then define when the notification should be sent.

Send notifications: [✓ Yes] No

Send notifications to: ARN [✓ New email or SMS topic]

Topic name (Name of the topic.): db-high-cpu-utilization-availability

With these recipients (Email addresses or phone numbers of SMS enabled devices to send the notifications to): x21168938@student.ncirl.ie

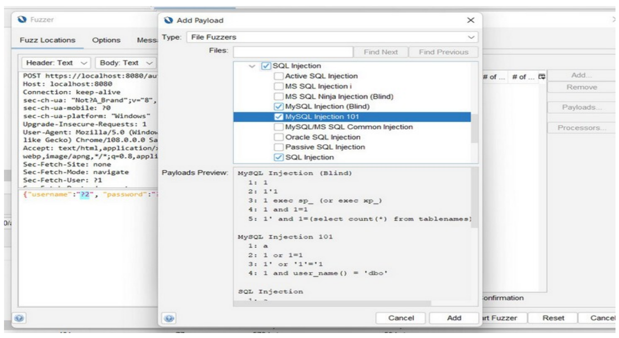
Metric: Average ▼ of CPU Utilization ▼

Threshold: >= ▼ 90 Percent

Evaluation period: 1 consecutive period(s) of 5 Minutes ▼

Name of alarm: awsrds-ca1-database1-mysql-High-CPU-Utilization

### A7 - SQL Injection payloads from ZAP



Type: File Fuzzers

```
POST https://localhost:8080/au
Host: localhost:8080
Connection: keep-alive
sec-ch-ua: "Not?A_Brand";v="8",
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Window
like Gecko) Chrome/108.0.0.0 Sa
Accept: text/html,application/
webp,image/apng,*/*;q=0.8,appli
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
```

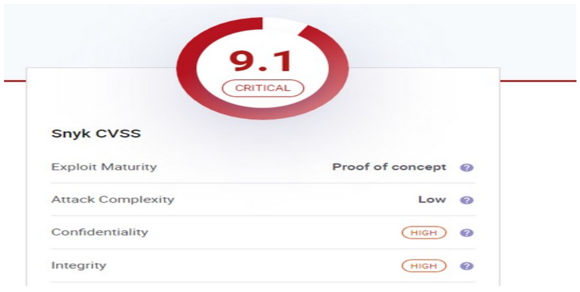
```
{"username":"?2", "password":"
```

- [✓] SQL Injection
  - [ ] Active SQL Injection
  - [ ] MS SQL Injection i
  - [ ] MS SQL Ninja Injection (Blind)
  - [✓] MySQL Injection (Blind)
  - [✓] MySQL Injection 101
  - [ ] MySQL/MS SQL Common Injection
  - [ ] Oracle SQL Injection
  - [ ] Passive SQL Injection
  - [✓] SQL Injection

Payloads Preview:

```
MySQL Injection (Blind)
  1: 1
  2: 1'1
  3: 1 exec sp_ (or exec xp_)
  4: 1 and 1=1
  5: 1' and 1=(select count(*) from tablenames)

MySQL Injection 101
  1: a
  2: 1 or 1=1
  3: 1' or '1'='1
  4: 1 and user_name() = 'dbo'

SQL Injection
```

## A8 - Snyk CVSS score for identified vulnerability



**Snyk CVSS**

| | |
|---|---|
| Exploit Maturity | Proof of concept |
| Attack Complexity | Low |
| Confidentiality | HIGH |
| Integrity | HIGH |

## REFERENCES

[1] Digital.AI, 16th Annual State of Agile Report, 2022, [Online] Available: https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf

[2] D.Gruber, Modern Application Development Security, August 2020, [Online] Available: https://www.veracode.com/sites/default/files/pdf/resources/surveyreports/esg-modernapplication-development-security-veracode-survey-report.pdf

[3]T.Gilb, Principles of Software Engineering Management(Wokingham, England: Addison-Wesley), 1988

[4] AWS Security and Compliance Guide, 2021, [Online] Available: https://d1.awsstatic.com/executive-insights/en_US/guide-security-compliance-quick-reference.pdf

[6] - AWS WAF FAQ, [Online] Available: https://aws.amazon.com/waf/faqs/#:~:text=AWS%20WAF%20helps%20protects%20your,that%20contain%20particular%20request%20headers.

[7] AWS Secrets Manager, [Online] Available: https://aws.amazon.com/secrets-manager/

[8] Amazon Inspector, https://us-east-1.console.aws.amazon.com/inspector/v2/home?region=us-east-1#/

[9] Amazon Inspector Features, [Online] Available: https://aws.amazon.com/inspector/features/?trk=a7f57dee-fc58-4084-9037-cb552d58a5d5&sc_channel=ps&s_kwcid=AL!4422!3!637214078350!!!g!!!19032308866!145290018913&ef_id=CjwKCAjwzuqgBhAcEiwAdj5dRhizFEsNN02ACE9UoSNEGmY-Uq_DF6nHL4ecOF1TB-aWJl9a1r1n1xoCTdMQAvD_BwE:G:s

[10] Tenable, Tenable.io and Amazon Web Services Integration Guide, February2023,[Online] Available: https:/ /docs.tenable.com/integrations/AWS/Content/PDF/Tenableio_and_Amazon_Web_Services_Integration_Guide.pdf

**Link to Presentation:** https://drive.google.com/drive/folders/1yipOETnaK2JkvwvXALHYkAC6rvMVDZr8?usp=share_link