DE MONTFORT UNIVERSITY LEICESTER

Development Project

Reducing Mean Time to Incident Response Through Automation

Final Deliverable

Project Author — Nathaniel Stevens (p2598625@mydmu.ac.uk)

Project Supervisor — Clinton Ingrams (cfi@dmu.ac.uk)

Code Base — Onedrive Folder

Word Count — 9 283 / 10 000

Note — The following sections have been carried forward from my first deliverable report

- Acknowledgments
- Functional Requirements
- Test Plan
- System Design

Table of Contents

Table of Tables

Table of Figures

Acknowledgements

Firstly, I would like to thank my Supervisor, Clinton Ingrams, for his constructive criticism, support and knowledge throughout the development of my first deliverable. I am very grateful for him agreeing to supervise my project and his input has been especially helpful.

Next, I would like to show appreciation to my employer, Matillion, whom without their support throughout my placement year and continued part-time employment during my final year, I would not of been able to develop this system. The training resources they have provided have been invaluable in both learning the concepts developed within the project but also helped build the prototype and provide an AWS account, free of charge.

Finally, I would like to especially thank both James Collinson and Karl Shrubb for taking the time to add their own testimonials to my project proposal document. I greatly appreciate their kind words and feedback on my initial plans.

Functional Requirements

I. Introduction

By identifying a system's users, intended functions can be mapped to their use-cases to ensure the application meets its objectives. Without this, the software development lifecycle is tunnel-visioned to just the desired usability of the developer.

II. Use Case Diagram

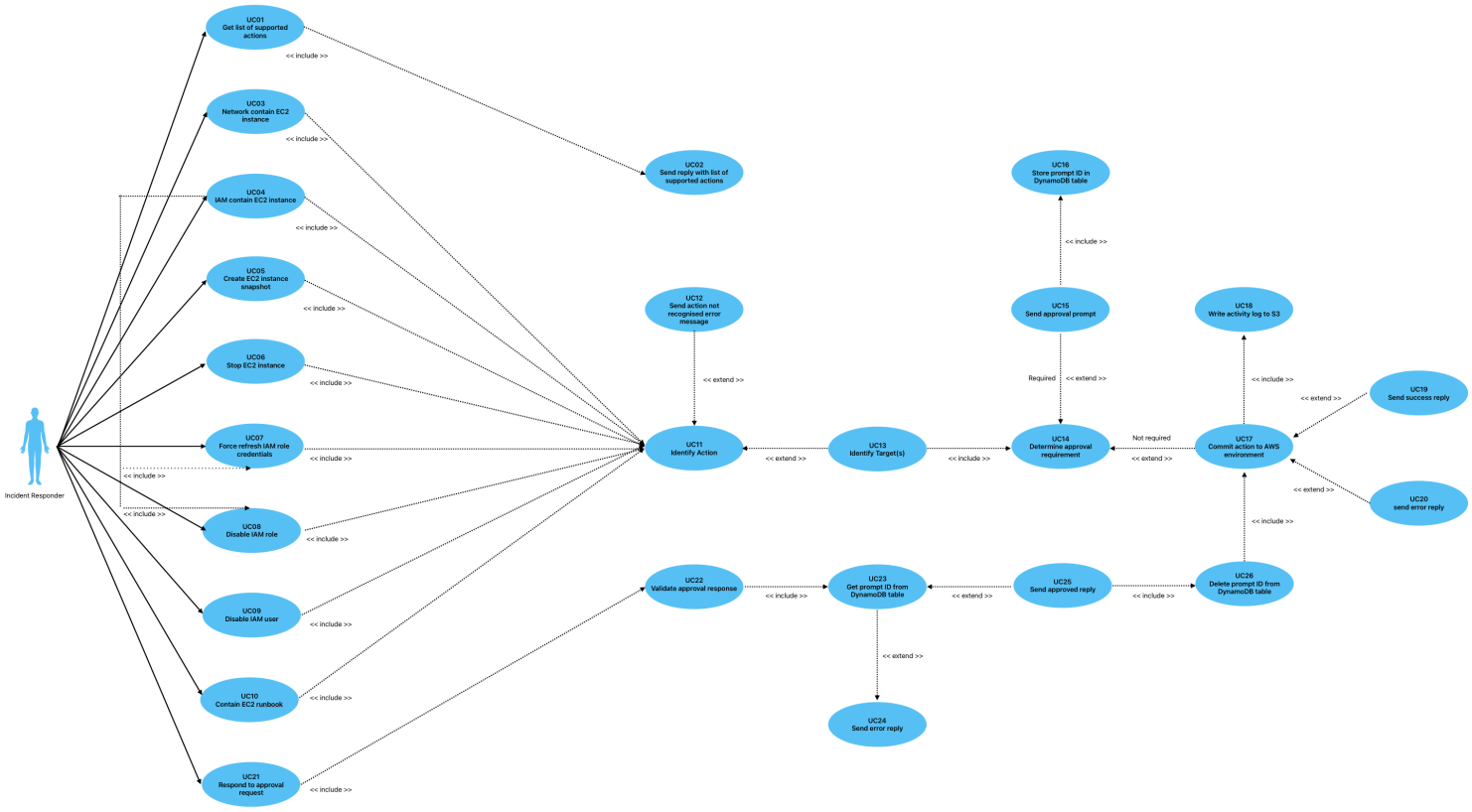

Figure 1. This diagram illustrates an actor's (user's) individual expectations by functions (use-cases). Additionally, any other use case that is included within the processing is noted, similarly, required functions that extend a given use case, if a set of conditions are met, are also shown (sparxsystems.com, no date).

III. Use Case Specification

Mirroring the use case diagram, this table specifies expected behaviour produced from a individual use case available to a user (sparxsystems.com, no date).

Furthermore, each case is defined through any constraining pre and post functions (sparxsystems.com, no date). These constraints specify any conditions that must be met either pre or post execution of the use case respectively.

Functional Requirements - Use Cases Table

Use Case ID	Actor	Use Case	Constraints	Requirements
UC01	Incident Responder	Get list of supported actions	• Pre: None • Post: UC02	Allows user to ask for a list of supported actions so they understand the capabilities of the system
UC02	System	Send reply with list of supported actions	• Pre: UC01 • Post: None	Informs user on system's capabilities with clear instruction on how to commit actions
UC03	Incident Responder	Network contain EC2 instance	• Pre: None • Post: UC11	Will restrict network access to a single / group of EC2 instances by removing the attached security group
UC04	Incident Responder	IAM contain EC2 instance	• Pre: None • Post: UC11	Will restrict a single / group of EC2 instances' API access to the AWS account by refreshing the attached IAM role's credentials (UC07) + applying deny-all permission policy (UC08)
UC05	Incident Responder	Create EC2 instance snapshot	• Pre: None • Post: UC11	Will take a full snapshot of a single / group of instances using Amazon Machine Image (AMI)
UC06	Incident Responder	Stop EC2 instance	• Pre: None • Post: UC11	Will stop a single / group of EC2 instances
UC07	Incident Responder	Force revoke IAM role credentials	• Pre: None • Post: UC11	Applies permission policy to a single / group of IAM roles to deny all actions where the credentials issue time was before action's time
UC08	Incident Responder	Disable IAM role	• Pre: None • Post: UC11	Applies permission policy to a single / group of IAM roles to deny all actions
UC09	Incident Responder	Disable IAM user	• Pre: None • Post: UC11	Applies permission policy to a single / group of IAM users to deny all actions and disables credentials
UC10	Incident Responder	Contain EC2 runbook	• Pre: None • Post: UC11	Commits UC03, UC04, UC05 and UC06 use cases together
UC11	System	Identify Action	• Pre: UC01, UC03:UC10 • Post: UC12 or UC13	Determines which action the user has included in their command
UC12	System	Send action not recognised error message	• Pre: UC11 • Post: None	Sends error message as reply, informing user the action they provided isn't recognised
UC13	System	Identify Target(s)	• Pre: UC11 • Post: UC14	Determines which target resource(s) the user has included in their command

Use Case ID	Actor	Use Case	Constraints	Requirements
UC14	System	Determine approval requirement	• Pre: UC13 • Post: UC15 or UC17	Determines whether action on asset requires approval from authorised list of users
UC15	System	Send approval prompt	• Pre: UC14 • Post: UC16	Sends reply, prompting for an authorised user to approve action on asset(s)
UC16	System	Store prompt ID in DynamoDB table	• Pre: UC15 • Post: None	Stores prompt ID in table as primary key with message ID and action as values
UC17	System	Commit action to AWS environment	• Pre: UC14 or UC26 • Post: UC18 and (UC19 or UC20)	Commits action on specified target(s) in AWS environment
UC18	System	Write activity log to S3	• Pre:UC17 • Post: None	Logs actions so user(s) can review incident response steps taken to recover to known-good state
UC19	System	Send success reply	• Pre: UC17 • Post: None	Informs user that response action has been completed
UC20	System	Send error reply	• Pre: UC17 • Post: None	Informs user that response action was not completed, provides error message from AWS
UC21	Incident Responder	Respond to approval request	• Pre: UC15 • Post: UC22	Allows a different user, to the one who triggered the action, to approve or deny an action by clicking a button
UC22	System	Validate approval response	• Pre: UC21 • Post: UC23	Ensures only an authorised AWS environment administrator has responded to prompt that isn't the same user who sent the original command
UC23	System	Get prompt ID from DynamoDB table	• Pre: UC22 • Post: UC24 or UC25	Using the Prompt ID as a primary key, retrieves the original message ID and action
UC24	System	Send error reply	• Pre: UC23 • Post: None	Informs user(s) that user who responded to prompt is not authorised to do so. Original prompt will continue to work
UC25	System	Send approval reply	• Pre: UC23 • Post: UC26	Informs users that user who responded to prompt was authorised and action is being committed
UC26	System	Delete prompt ID from DynamoDB table	• Pre: UC25 • Post: UC17	Removes data related to prompt ID from table as it is no longer needed

Test Plan

I. Strategy

Tests will be broken down into individual cases to identify the specific objective and whether the application meets this goal. To ensure a rapid feedback loop from test results to development and improvements, test cases will be evaluated as the features within the app are created. Therefore, in the first deliverable, some tests will not have a result as the feature is yet to be developed.

All tests will be mapped to at least one functional requirement and have an associated priority based on their contribution to meet the business requirement of the system. Using this metric, development work to fix failed tests can be allocated where it will produce the most value. Any re-tests after fixes will be noted with their final result of pass / fail.

II. Objectives

Through regular testing when features are developed, the aim of tests is to ensure the business requirement of the system is met. If left to the final product, a long period of time would have passed between the start and end of development with the solution likely not meeting the stated requirements.

Furthermore, the tests will highlight any deficiencies in the original system design and allow for improvements where time and value permits.

Overall, an application that meets all stated functional requirements and satisfies the business requirement of reducing the mean time to incident response through automation will be deemed a successful solution developed and objectives met.

III. Cases

Note —

- Text in backticks, “ ` ”, denote messages sent in slack channel to command automation
- The table has now been completed in this report since the first deliverable as the later case features have been developed

Test Plan - Test Cases Table

Test Case ID	Use Case ID	Objective	Procedure	Expected Result	Actual Result	Re-test Result (If needed)
TC01	UC01 UC11 UC02	Send command to bot to get list of defined supported actions	`@bot help`	Reply with success response with list of supported actions	Reply with success response with list of supported actions	N/A
TC02	UC01 UC11 UC02	Send command to bot to get list of supported actions when there are none defined	`@bot help`	Reply with error response stating no defined supported actions	Error in code. Logic Expected dictionary but got string	Reply with error response stating no defined supported actions
TC03	UC03-10 UC11 UC13 UC14 UC17 UC19	Send command to bot to perform incident response action that doesn’t require approval	`@bot “action” “instance ID” “region” “Account ID”`	Reply with Success response	Reply with Success response	N/A
TC04	UC03-10 UC12 UC13 UC14	Send command to bot to perform incident response action that does require approval	`@bot “action” “instance ID” “region” “Account ID”`	Reply with approval prompt	Reply with approval prompt	N/A
TC05	UC03-10 UC11 UC13 UC14 UC17 UC20	Send command to bot to perform incident response action on a non-existent target	`@bot “action” “instance ID” “region” “Account ID”`	Reply with error response, stating instance not found	Reply with error response, stating instance not found	N/A

Test Case ID	Use Case ID	Objective	Procedure	Expected Result	Actual Result	Re-test Result (If needed)
TC06	UC11 UC12	Send command to bot that isn't recognised	`@bot iam-contain-lambda "function ID" "region" "Account ID"`	Reply with error response, stating action requested isn't recognised	Reply with error response, stating action requested isn't recognised	N/A
TC07	UC21 UC22 UC23 UC25 UC26 UC17 UC19	Respond to approval request as an authorised user	Click "Approve button" whilst logged in as an authorised user	Approval determined valid, replies with success message and originally requested action is committed	Error in code. User who requested action could approve	Approval determined valid, replies with success message and originally requested action is committed
TC08	UC21 UC22 UC23 UC24	Respond to approval request as a non-authorised user	Click "Approve button" whilst logged in as a non-authorised user	Approval determined invalid, replies with error message	Error in code. User who requested action could approve	Approval determined invalid, replies with error message
TC09	UC01 UC03-10	Send command to bot using natural language where the target doesn't require approval	`@bot we need the IAM role "role name" disabled in "Account ID" to prevent it being able to make any actions`	Extract the action, target and AWS account from the prompt and reply with success response	Response from ChatGPT did not reliably produce the correct answer	Extract the action, target and AWS account from the prompt and reply with success response
TC10	UC01 UC03-10	Send command to bot using natural language where the target does require approval	`@bot we need the IAM role "role name" disabled in "Account ID" to prevent it being able to make any actions`	Extract the action, target and AWS account from the prompt and reply with approval prompt	Extract the action, target and AWS account from the prompt and reply with approval prompt	N/A

Test Case ID	Use Case ID	Objective	Procedure	Expected Result	Actual Result	Re-test Result (If needed)
TC11	UC01 UC03-10	Send command to bot using natural language where the target doesn't exist	`@bot we need the IAM role "role name" disabled in "Account ID" to prevent it being able to make any actions`	Extract the action, target and AWS account from the prompt and reply with error response, stating IAM role not found	Extract the action, target and AWS account from the prompt and reply with error response, stating IAM role not found	N/A

System Design

I. Introduction

For the entirety of the development process for this project, the 12 Factor App methodology (Wiggins, 2017) has been followed to effectively build a software-as-a-service application, suitable for deployment in a cloud-native environment. Therefore, the twelve factors will be referenced to highlight alignment with the methodology. Meanwhile, Due to budgetary constraints, principals such as "Keep development, staging, and production as similar as possible" were not entirely possible as separate environments could not be built for testing and production.

II. Infrastructure

Summary

Resources that formulate the infrastructure upon which the system runs is built and managed using code. Programmatically orchestrating infrastructure provides the benefit of reproducible results for each instance that the eco-system is built. Likewise, it is less error-prone as duplicated tasks can be conducted in loops rather than repetitive actions in which humans can develop fatigue. Moreover, the system is highly portable, as long as the infrastructure-as-code is available, any administrator can replicate the environment and utilise the system.

Diagram

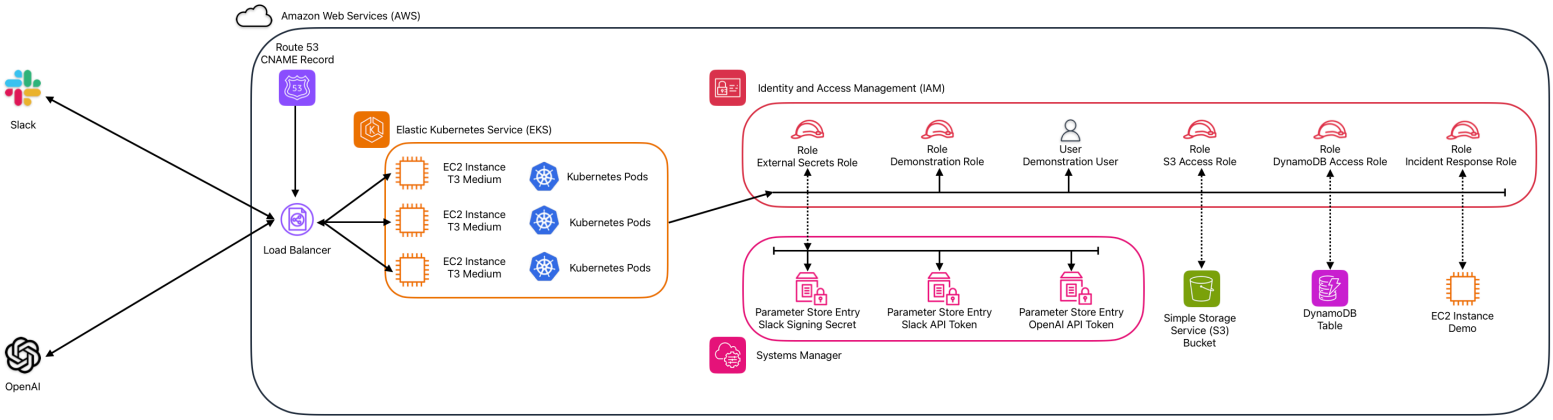

Figure 2. Diagram shows high-level layout and relationships between resources created within AWS account along with the connected external services Slack and OpenAI.

Slack

This third party application will be the the interface by which users interact with the system. Messages can be sent into a channel, commanding incident response actions to be conducted in the AWS Account. A Slack app will then forward users' messages and interactions to the system where they will be processed and a response shown as a reply to the original message.

The Slack app is given permissions to conduct activity within the workspace. Through Application Programming Interface (API) requests, the system can programmatically send and receive information from the Slack channel and the member users. (See Appendix A for bot's configuration).

OpenAI

As an extension to the basic feature of sending declarative commands (See Figure 3), users will also be able to write natural language prompts (See figure 4) which will be processed by a Large Language Model to extract required information like the action and target information.

Load Balancer

This will forward HTTP traffic to the appropriate container, running on one of the worker nodes.

Elastic Kubernetes Service (EKS)

Kubernetes is a container orchestration platform with extensive open-source projects that provide extensions to the core capabilities. EKS is Amazon's managed service in which they are responsible for the control plane and the customer manages the data plane.

There are no persistent data storage services in the cluster used by the system. This is to ensure the environment remains stateless and stores any data in a remote backing service (DynamoDB and S3). With the possibility of failures and dynamic scaling of Knative services, storing data within the cluster would not be reliable or stable.

Within the cluster's data plane, multiple services will be deployed to provide extra functionality which is not offered natively. These include:

- **Knative** for scaling containers to zero instances and dynamically scaling horizontally by receiving HTTP traffic. This will allow the system to serve messages from the Slack channel as they are forwarded via HTTP events. Each knative service has a release number to maintain strict separation between the build, release and run stages.

- **Cert Manager** will automatically provision Transport Layer Security (TLS) certificates within the cluster to internet-exposed Knative services so traffic sent between slack and the system is encrypted.

- **External Secrets** will maintain a synchronised state between the Kubernetes cluster's secret resources and the Parameter Store entries in AWS System's Manager. It is not best practice to store secrets, for example API tokens, in a code-base therefore a managed secret storage solution is used as the source of truth.

- **Linkerd** is a service mesh which will be used to provide mutual TLS (mTLS) between all internal network traffic. This is achieved by deploying proxies next-to each service which in-turn encrypts traffic using HTTPS (Morgan, No date).

DynamoDB

A simple key-value, no Structured Query Language (SQL), storage solution offered by AWS where there is no compute resources to maintain by the customer. Data retrieval is done through simple API requests, referencing a hash key (AWS, no date - a).

Identity & Access Management (IAM)

It is important to create a strict environment of least-privilege to avoid abuse of resources and data in a cloud environment. Through roles, permissions can be delegated to resources but also assumed by users for a set period of time, after which re-authentication is required.

For demonstrating the capabilities of the system, a user and role will be created which will have set permissions of what they can do. However, IAM users' credentials can be used to interact with an AWS account through a Graphical User Interface (GUI) known as the console and do not expire.

Systems Manager - Parameter store

A more financially efficient alternative to AWS' Secrets Manager, entries can be stored as secrets which are encrypted at rest and conditional access can be set to restrict who can read the values.

The following secrets will be stored:

- **Slack Signing Secret** used to validate received events are from Slack
- **Slack API Token** used to authenticate to Slack's API
- **OpenAI API Token** used to authenticate to OpenAI, allowing prompts to be programmatically sent
- **Docker Registry token** used to authenticate to private docker registry which stores container images

Simple Storage Service (S3)

Similarly to DynamoDB, S3 is a managed storage solution but differs by storing files. Every action conducted by the system within the AWS account will be logged and stored as a file in an S3 bucket. The purpose of this is to allow incident responders to audit the actions they took during an incident and determine what changes were made so returning to a known-good state is more easily achieved.

III. Database Schema

System Design - Database Schema - Jarvis Table

Column Name	Data Type	Description
callback_id	**string**	**Unique identifier for approval prompt. Used to retrieve an entry after a response is received**
service_name	string	Name of service that used requested. This is synonymous with the action's name
aws_account_id	string	Unique identifier for AWS Account that user set in their prompt
aws_region	string	AWS Region name that user set in their prompt. If the region isn't needed (E.g. For IAM actions), this is set to "global"
Targets	String	Comma separated list of target assets to commit action against

Note — The table's primary / hash key is formatted in **bold**

Only one table will be used to store persistent data for the purpose of remembering the actions a user has asked to be committed but require approval. Once approval has been given the associated data will be retrieved from the table. The callback_id key will be the primary key / hash key used to get data as this is included in the body of any approval response. The service_name references which action the user wishes to commit. Furthermore, associated data regarding the AWS environment the action is to be committed in is stored (aws_account_id and aws_region). The targets are the resource IDs to be actioned upon.

No sensitive data will be stored in this table and once an approval request has been successfully validated, the entry will be deleted. Additionally, a time-to-live attribute will be assigned to each entry so old (>60 days) data is automatically deleted (AWS, no date - b)

As this is the only table to be created and used, there is no relationships to model with other tables / databases.

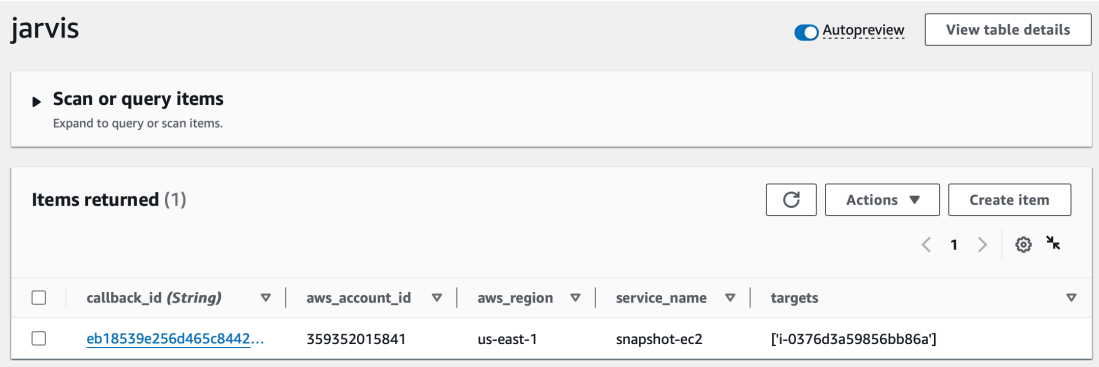

Figure 3. Data entry stored in DynamoDB for an approval request.

IV. User Interface

Traditional Command Prompt

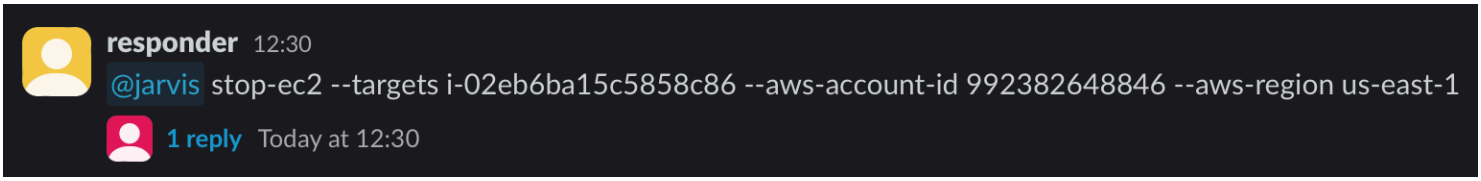

Figure 4. Shows user sending command to bot via declarative command.

Natural Language Command Prompt

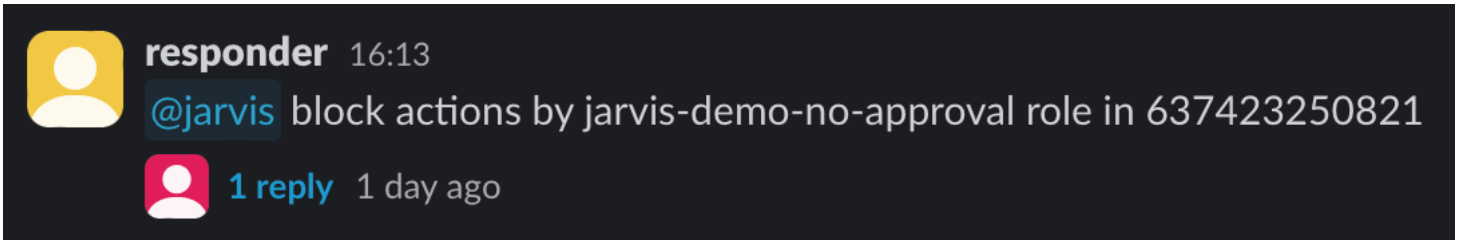

Figure 5. Shows user sending command to automation using natural language.

Success Response

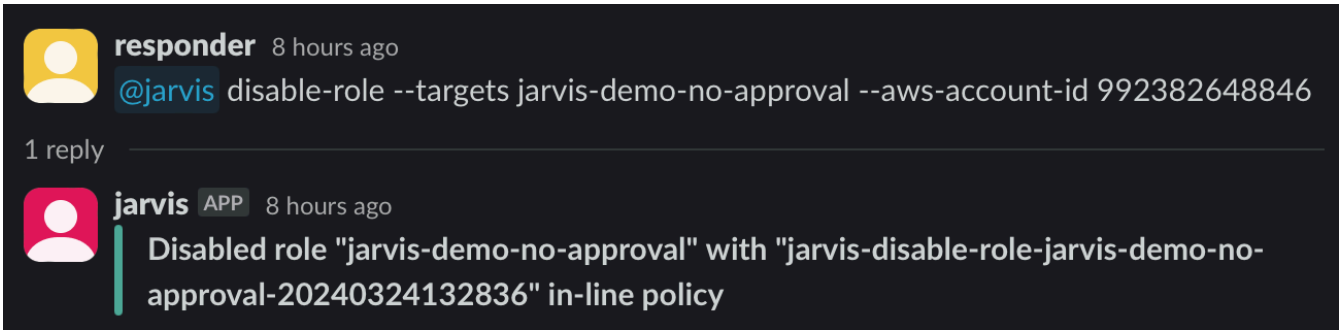

Figure 6. Shows automation bot responding with list of supported actions and their description.

Error Response

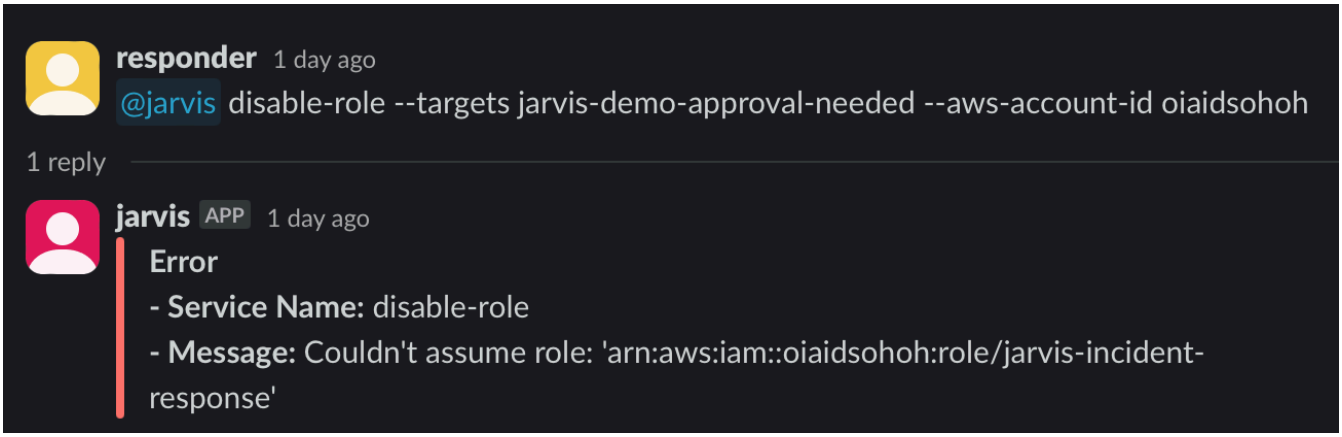

Figure 7. Shows automation responding with an error message due to the command not being recognised.

Approval Request

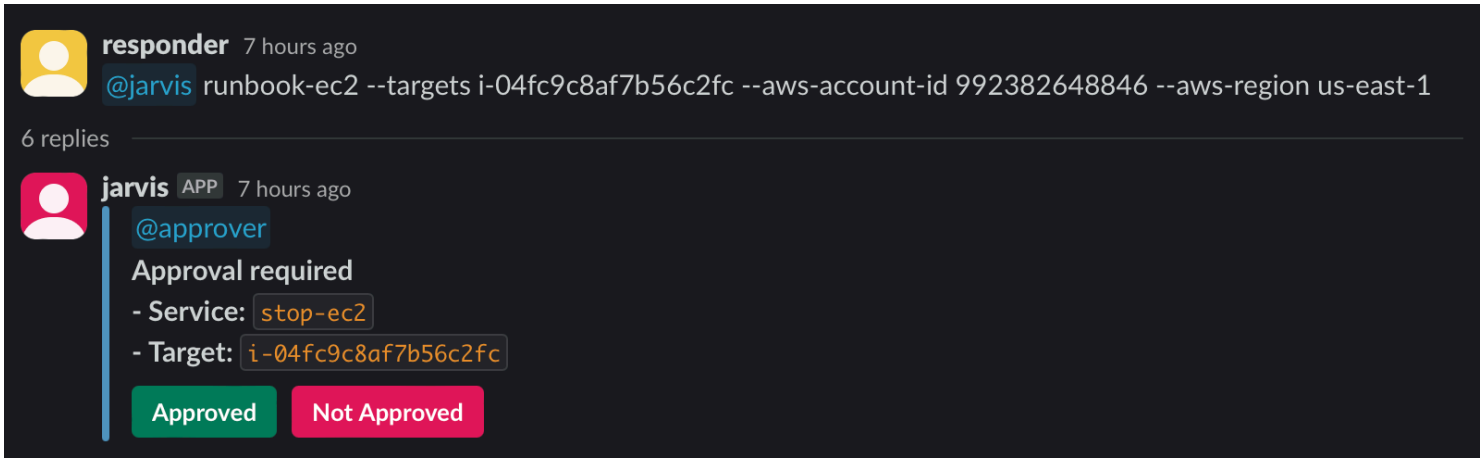

Figure 8. Shows automation responding with an approval request. The person who is authorised to approve the request has been mentioned in the message for clarity ("@approver").

V. Application Code

Primarily, the logical system is a group of HTTP servers written in Python and exposed via port binding to the cluster's local network. Slack events and interactions are directed to central, internet-accessible services called "switchboard" which route traffic to the relevant microservice based on the command prescribed by the user. Helper services for common tasks are containerised into individual applications to reduce duplication of code.

Python was chosen as the programming language due to its extensive popularity in the software development community. My predominant programming knowledge is with Python and so I am most comfortable writing code in this Syntax.

Configuration values for the microservices will be stored in Kubernetes Configmaps to keep configuration values close to the application but also easily changeable if needed.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: disable-role

data:
  require_approval: "false"

  approvers: |
    02tins.seeker@icloud.com

  arguments: |
    "--targets"="list"
    "--aws-account-id"="string"
```

Figure 9. Example Configmap for the disable-role action. This defines whether the action needs approval before it can be completed, the authorised users to approve and the expected arguments to be passed to the service with their associated types.

No secrets are stored in code as this goes against best practices (Jackson, 2023) and therefore are in a secret store within AWS (Parameter Store), synced to Kubernetes Secret resources via External Secrets and included into containers via the native SecretKeyRef functionality as environment variables.

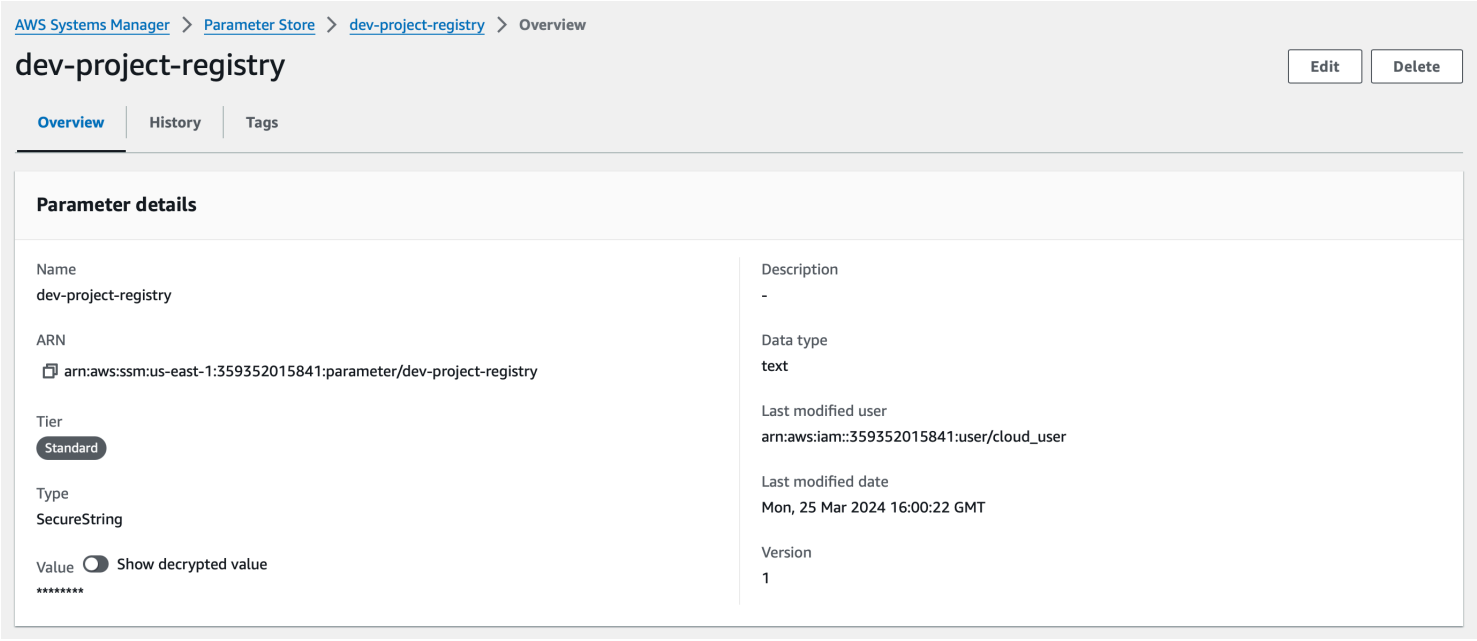

Figure 10. Example secret stored in AWS' Systems Manager Parameter Store.

```yaml
apiVersion: external-secrets.io/v1beta1
kind: ClusterSecretStore
metadata:
  name: aws-{{ .Values.aws_region }}-parameter-store

spec:
  provider:
    aws:
      service: ParameterStore
      region: {{ .Values.aws_region }}
      auth:
        jwt:
          serviceAccountRef:
            name: parameter-store-read
            namespace: external-secrets
```

Figure 11. YAML code for External Secrets Secret Store. I am using the Cluster scoped version so secrets can be created in any namespace. "{{ .Values.aws_region }}" references a variable in the External Secrets' Helm chart which defines the AWS Region in which the Parameter Store values are located. Finally, to authenticate to AWS' Parameter Store, I am using a service account ("parameter-store-read") which can assume an AWS role which has the necessary permissions to read the parameter values.

```yaml
apiVersion: external-secrets.io/v1beta1
kind: ClusterExternalSecret
metadata:
  name: "dev-project-registry"

spec:
  externalSecretName: "dev-project-registry"

  namespaceSelector:
    matchExpressions:
    - { key: external-secrets-target, operator: In, values: [slack, jarvis, aws] }

  refreshTime: "10m"

  externalSecretSpec:

    refreshInterval: "10m"

    secretStoreRef:
      name: "aws-{{ .Values.aws_region }}-parameter-store"
      kind: ClusterSecretStore

    target:
      # Name of kubernetes secret
      name: dev-project-registry

      creationPolicy: "Owner"
      deletionPolicy: "Delete"

      template:
        type: kubernetes.io/dockerconfigjson
        data:
          .dockerconfigjson: "{{ `{{ .secret }}` }}"

    data:
        # Key in kubernetes secret
      - secretKey: secret
        remoteRef:
          # Name of paramater store paramater
          key: dev-project-registry

          conversionStrategy: Default
          decodingStrategy: None
          metadataPolicy: None
```

Figure 12. YAML code for Kubernetes secret resource that will be created via External Secrets. This resource, called “dev-project-registry” will be created in namespaces that match the list of label selectors with a key of “external-secrets-target” and values of “slack”, “jarvis” or “aws”. The Kubernetes secret resource will be synced with the truth-of-source Parameter Store value every 10 minutes.

All dependencies are referenced in the Dockerfiles which are used to define the containers. Using a minimal base python image, only the required packages are included therefore keeping the containers small in size and easy to deploy.

```
FROM python:slim-bullseye

USER root

# Install dependencies
RUN apt update -y

# Create new user
RUN adduser --disabled-password autouser

# Copy over scripts
COPY server.py /home/autouser/scripts/server.py
COPY service_helper.py /home/autouser/scripts/service_helper.py
COPY read_configmap.py /home/autouser/scripts/read_configmap.py
COPY action.py /home/autouser/scripts/action.py

# Set permissions
RUN chown autouser:autouser -R /home/autouser/scripts
RUN chmod +x /home/autouser/scripts/*.py

# Install python modules
RUN pip3 install requests \
    && pip3 install boto3

# Change user
USER autouser

# Change directory
WORKDIR /home/autouser/scripts

# Run script on start
CMD [ "python3", "server.py" ]
```

Figure 13. An example Dockerfile which defines a container image. This example will utilise the python base image with the version name “slim-bullseye”. The image will be updated, a new “autouser” user will be created and local files will be copied onto the container. All said files will be configured to be owned by the newly created user and python libraries will be installed too. All of which will run as the root user due to the need of elevated permissions. After which, the user can be changed to “autouser” and set the working directory to that user’s home folder. Finally, when the container starts, it will run the “server.py” python script.

Each microservice logs events to stdout and can be read via a Kubectl command. This provides time ordered visibility to the apps' behaviour and easy troubleshooting of errors. A log of actions conducted in the AWS account are stored in S3 for long-term access.

```
[+] Server Started: localhost:5555
[+] Verification Event Receieved ...
 | Verified

--- 2024-03-25 16:28:30 | Received POST request ---

[+] Authenticating Event ...
 | Slack timestamp: 1711384110
 | Local timestamp: 1711384110
 | Slack Signing Secret: f1f2b...
 | Slack signature: v0=e74596084dc57a37d2f51ded11305998398e57f1684fd66eeb29ebfae0f4feab
 | Local signature: v0=e74596084dc57a37d2f51ded11305998398e57f1684fd66eeb29ebfae0f4feab
 | Signatures match, authentic

[+] Configuration Values
 | bot_id: U0676P8JKEH
 | commands.mapping: {'help': 'http://help.jarvis.svc.cluster.local', 'stop-ec2': 'http://stop-
ec2.aws.svc.cluster.local', 'network-contain-ec2': 'http://network-contain-ec2.aws.svc.cluster.local', 'iam-
contain-ec2': 'http://iam-contain-ec2.aws.svc.cluster.local', 'snapshot-ec2': 'http://snapshot-
ec2.aws.svc.cluster.local', 'refresh-user-session': 'http://refresh-user-session.aws.svc.cluster.local',
'refresh-role-session': 'http://refresh-role-session.aws.svc.cluster.local', 'disable-user': 'http://disable-
user.aws.svc.cluster.local', 'disable-role': 'http://disable-role.aws.svc.cluster.local', 'runbook-ec2':
'http://runbook-ec2.jarvis.svc.cluster.local'}

[+] Event Details
 | Channel ID: C06BX2YSRLK
 | Thread ID: 1711384109.714919
 | User ID: U06BVM9K7K8
 | Event Type: app_mention
 | Event Text: <@U0676P8JKEH> snapshot-ec2 --targets i-0376d3a59856bb86a --aws-account-id 359352015841 --aws-
region us-east-1

[+] Handling App Mention Event ...
 | Command: snapshot-ec2
 | Prompt: --targets i-0376d3a59856bb86a --aws-account-id 359352015841 --aws-region us-east-1
 | Sent to http://snapshot-ec2.aws.svc.cluster.local/event
```

Figure 14. Example log output from the events-switchboard service. This shows a received event being authenticated, configuration values read from the attached Configmap and the important event's details identified. Next, the switchboard determines and sends the event to the appropriate service.

Major Components

I. Problem Area

The problem my project aims to solve is to minimise the need for manual response to threat detections, therefore decrease the mean time to incident response. Similarly, by codifying actions, they can be documented, agreed on with stakeholders and tested during exercises to ensure they meet expectations. Finally, the system will integrate with already popular tools used by businesses such as AWS' cloud and Slack's messaging application.

Within cloud estates, vast number of assets exist and often scale dynamically to meet demand (Check Point, no date). Due to this nature, manually changing configurations to prevent the spread of a compromise will not scale within an adequate response time. Furthermore, humans often experience fatigue from repetitive actions (Atlassian, no date - a) and so it is likely that an error will occur if the same method of containment is applied to multiple targets.

A requirement to manually intervene within a cloud environment is privileged access. The principal of least privilege dictates that people should have the minimal level of access they require to complete their job (Microsoft, 2023). To satisfy this, security personnel will need permissions to modify assets in production environments and this can be achieved in multiple ways. Firstly, permissions can be given when needed which will reduce the impact of a user account being compromised but will slow response. Secondly, the responsibility can be shifted entirely to another team, perhaps individuals who already have such access for other needs. This would again reduce the impact of a security user's account being compromised but will slow response and prevent security personnel from potentially making breaking changes without the full awareness of the consequences but also requires more people involved. Lastly, the privileges can be pre-allocated so they are always available when needed. The main concern of such is that if a threat actor were to gain access to their account, they would have high levels of access to sensitive systems.

To offset the concerns outlined above, privileges can be assigned to automation which only conducts specific actions defined in code. Not only does this allow for security personnel to conduct their incident response plan in a timely manner but appreciates infrastructure and development team's desire to pre-approve response actions, preventing any breaking changes that don't need to be made as pull requests in code repositories can be used. This is the direction my project takes, response actions outlined in a plan can be codified, collaborated on with other teams to ensure the best approach is being taken via pull requests then stored in a central repository for automation to utilise when triggered.

Once an organisation begins developing silos of their teams, developers, infrastructure and security to name a few, it can be easy to work within a vacuum and not collaborate with others due to their own priorities. To automate actions, they need to be written in code and this empirical document of process can be shared with others for approval. Technical engineers will be able to understand the detailed nuances and implications of the

proposed actions which can be filtered and passed up to operational stakeholders who can be assured that a sound, repeatable step of actions will be followed every time with known results. Traditional playbooks that only detail high-level responses do not provide this clarity and assurance but leave room for interpretation by the incident responder which can lead to misunderstandings and consequences to the availability of production systems.

Likewise, similarly to software development for products, once code is produced tests can be carried out to ensure it meets expectations and confidence be instilled in the security team's intentions and abilities. It is paramount that any incident response plan is stress tested so issues can be identified and remediated before they are used in reality. As mentioned before, the high-level nature of traditional playbooks can vary the tests conducted by incident responders due to how they interpret the steps outlined within. By moving these into code, the same process will be followed every time and so the prerequisite requirements can be confidently defined to ensure errors don't occur.

Companies looking for ways to automate their incident response process do not wish to change their existing tooling that is intended to integrate with such a system. Platforms that detect threats and those used to communicate take lots of resources to migrate away from and therefore it is important that compatibility is available either out-of-the-box or possible to be added throughout a system that would sit in between others. My system integrates with the popular AWS cloud provider (Statista, 2022) and is triggered by and outputs to a slack channel which are widely used in organisations for communication (Slack, 2019).

With the growing popularity of Artificial Intelligence (AI), features that utilise such capabilities have been quickly adopted without much research into their actual benefits and uses (Power, 2024). Without a clear goal for what the AI will bring to the system, their presence can cause more harm than good due to generative model's tendency to produce incorrect information (Niekerk and Williamson, 2023). On the other hand, I have identified a useful area in which generative AI can improve a user's experience of incident response automation. Instead of being required to follow a set format in command structure, users can write natural language prompts with the system deciphering this, finding the useful required pieces of information. This not only makes the tool easier to use, it will also lower the knowledge requirements of incident response as users can request for an action without needing to know the technical steps to do it.

II. Objectives

All objectives of this project have the final aim to reduce the time of response to a security incident. The three main objective areas are as follows, a user can interact with the automation within an already widely used 3rd party platform (Slack), The automation can conduct actions within an AWS account and empirical evidence is detailed within this report to demonstrate how the time to incident response has reduced.

Through interacting with the automation via Slack, multiple people can view response actions, collaborate and discuss on the best steps to take whilst not needing to use a separate system that they aren't experienced with.

The automation should be able to intake and understand commands written in either an explicit format (see Figure 3) or via natural language prompts (see Figure 4). Furthermore, the system should inform users when an action has been successfully completed, when approval is required or when an error has occurred.

This project focusses on containment actions for incident response taken within an AWS account. The actions written will only work from a technical standpoint within said Cloud Service Provider (CSP) however, on a high level, the same process could be followed to implement such processes within other CSPs such as Azure and Google's Cloud Platform (GCP). Any errors that prevent an action to be taken within the account should be relayed back to the user so they can make an informed decision on how best to remediate. Likewise, when the state of a target / targets has been modified successfully, the user should be informed so they can share with others that the threat has been contained. Logs of state changes should be created within persistent storage (S3) so incident responders can later return assets to their previous state before compromise (with adequate hardening measures implemented in addition to prevent another compromise however this aspect is outside the scope of this project).

Finally, to demonstrate the successfulness of the system, evidence will be calculated and provided to show how the automation reduces the time to respond to a security incident via containing the threat (see Critical Analysis & Reflection - Appraisal of Final Product - Incident Response Times table). This will be done by firstly measuring the response time via manual means using the graphic console. After, the automation will be triggered to conduct the same steps and measured from time of prompt to success message. Finally, a conclusion can then be drawn on whether the system reduces the time to respond and if any caveats exist for said objective.

III. Rationale For Design & Implementation Decisiones

For the design of the system in this project, a popular methodology called the 12 Factor App was followed to meet industry standards of a software-as-a-service (SaaS) application. The rationale for using this methodology over others is predominantly due to its detailed requirements outlined and its examples of contradictions to the factors. This clear documentation made it easy for me to meet industry standards for creating an application.

The first factor is tracking the application's code in a central revision control repository. For this project, GitHub was used similarly to the reasons stated before; its popularity and my previous familiarity with it. The purpose of a central repository is to maintain a source of truth for the application from which developers can build upon in their local development environment and multiple deployments can be facilitated to different environments such as production for customer interaction and staging for testing. However, for this project, due to budgetary constraints, these separate deployment environments were not possible so only one was created and used.

Next, dependencies should be explicitly declared within a manifest and not to rely on packages being installed as a bundle with the underlying operating system. The benefits of this are to ensure the applications portability across environments as only only the codebase, programming language's runtime and associated dependency manager are required and can be built using automation. Furthermore, a complete list of dependencies improves security as versions can be easily updated following new releases so vulnerabilities are not included within the running application. These requirements have been facilitated in this project via declaring dependencies in the Dockerfiles used to create each containerised service of the application. Both Python libraries and system tools are installed via package managers, pip and apt respectively. In-turn, these Dockerfiles are stored in the codebase so the containers can be easily built using docker.

Following from this, configuration values that vary between deployments should be stored within the environment rather than the codebase as constant variables. This strict separation of config from code makes deployments to different environments (as mentioned earlier, production and staging as examples) more efficient as the codebase can remain the same. Further to this, security is improved as credentials are not stored in the codebase which would go against Github's best practice recommendations (GitHub, no date). This project's system complies with this factor by storing configuration values in Kuberntes configmaps and secrets in Kubernetes secrets. The former is mounted as a file onto running containers and then read by the application code to configure at runtime. The latter is mounted as environment variables and similarly read by the application code at runtime and injected into API calls to external services. Additionally, the Kubernetes secret resources are synced with an external secret storage service provided by AWS called Secrets Manager which provides granular access control and encryption. This synchronisation is facilitated by the External Secrets Operator in Kubernetes as it has native support for AWS Secrets Manager.

Moving on to the fourth factor, backing services consumed and interacted with by the application should be treated as attached resources. No distinction between 3rd party services and local ones should be made with both being accessed via a Uniform Resource Locator (URL). The meaning of "attached resource" is to say that the services are loosely coupled with the core application so they can be changed at will with no alterations to the codebase needed. The URLs used to interact should be stored as configuration values as discussed previously. In the context of the system developed for this project, backing services are resources within AWS, Slack's API and the local containerised micro-services which conduct specific actions. All interactions with AWS are facilitated through a python library called boto3 and therefore no URLs are needed. Slack's API and the microservices' local URL are stored in Kubernetes configmaps, mounted as files in containers and read at runtime by the application. No changes to the application code would be needed if a backing service's URL was to change.

Separate stages of deployment should be followed to transform the codebase to a running application. Firstly, the build stage creates an executable of the app code stored in version control with all the defined dependencies included. Secondly, the release stage combines the executable with the configuration values for immediate execution. Thirdly, the run stage executes the combined executable and configuration in the deployment

environment. The advantages of this staged approach is to prevent changes at runtime propagating down-stream without being checked into version control and peer-reviewed. Also, each release should have a unique identifier to measure progress but also to facilitate rollbacks incase of an unseen breaking change. Each stage is implemented within this project's system by firstly transforming the application code to executable container images using docker. These are then combined with the configuration values stored in Kubernetes configmaps within the knative service manifests and finally deployed to the runtime environment via ArgoCD which ensures the resources defined in GitHub are reflected in the Kubernetes cluster's state.

For execution, the application should be stateless and be ran as one or more processes. By not storing any data locally (stateless), an app failure will not result in data loss which could mean a user losing their information. Instead, data is stored in a backing service which requirements are detailed in the fourth factor. To ensure efficient scaling of the application to meet user demands, running as processes means as demand increases, more processes of the same app can be ran to satisfy. The system for this projects is stateless and stores no data locally when running. All information is either stored within a database service called DynamoDB or an object storage service called S3 both of which are managed and provided by AWS. Moreover, containers are, at their fundamental level, processes running on the host system (McCune, 2023) (in this case the Kubernetes cluster's nodes). Therefore, as demand increases, Knative will dynamically scale the number of running containers to serve the traffic received.

Local web services that form the application as a whole should be exported via port binding. This entails specifying a unique port number to use so network traffic can be received and passed to the specific service. This approach provides easy HTTP access to each service from others with the URL for each being defined in other's configuration where needed. The basic principal of Knative is to scale containers based on the volume of HTTP traffic received and therefore makes port binding easy to implement. Within each Knative service's manifest, a port number is defined for the container which will receive any HTTP traffic sent to it.

Building upon factor six, using processes as the execution form of the system allows for efficient horizontal scaling. As demand increases, the number of concurrently running containers will increase to satisfy the demand. Furthermore, each workload should be an individual process so compute resources are dedicated to the most in-demand aspect of the application. Within this project, as mentioned before, the system runs as a web server process in containers and can horizontally scale as HTTP traffic to each individual service increases.

Due to the stateless nature of the application, all running services are treated as cattle, not pets. Because of this, each is disposable which is the cornerstone of factor nine. With minimal startup time, scaling can respond to demand rapidly with minimal lag. Likewise, graceful shutdowns via ceasing to listen on the designated port will complete execution on current data however prevent any new from being received and served. This is a native feature of Knative in which containers are terminated when HTTP traffic is no longer received.

For the tenth factor, development and production deployment environments should be be as similar as possible in architecture, deployment method and available backing services. As mentioned before, separate environments were not possible in the development of this project however the concept of continuous delivery is facilitated in the project using GitHub as the source of code and ArgoCD to maintain sync. Using these, any new code can be deployed and running within the Kubernetes cluster in minutes.

It is crucial for an application to effectively log activity so errors can be identified and remediated quickly. To this end, factor eleven dictates that logs should be streamed from beginning to end of a services' runtime in chronological order. The benefit of this if to reduce the need of managing log files which, when large, utilise high amounts of storage and violate the sixth factor of not being stateless. Additionally, it is easy to read the log output and can be exported to an archival storage backing service external to the application. For this project, logs are sent to stdout using the Python logging library with any errors both highlighted here and also sent to Slack for the user to be made aware of.

Finally, the twelfth factor mandates admin tasks to be ran as one-off processes in identical environments. Any admin code should be shipped with application code to avoid synchronisation issues and similar dependency management used in factor two should be used as well. For this project's application, administration tasks can be executed by establishing a shell on a running container using the exec Kubectl command.

Development Lifecycle

I. Software Development Methodology

Due to the nature of this project and the fact I was the sole contributor (although online resources were of great help and are referenced throughout this report), I chose to follow the DevOps software development methodology. This focuses on combining the development and operations functions to automate, get fast feedback and iterate on product delivery (GitLab, no date; IBM, no date).

The first core principal of DevOps to guide effectiveness is automating the software development lifecycle. This reduces the time it takes for code to be deployed within an environment and removes the aspect of human error from repetitive tasks (GitLab, no date; IBM, no date; Atlassian, no date - b).

Next, improvements on the iterative cycles should be continuous so waste is minimised. Automation goes a long way to achieve this but learning from previous tests will identify areas that can be improved from both a technical but also user experience perspective (GitLab, no date; Atlassian, no date - b).

The final core principal is to focus on user's needs as without their satisfaction, the developed product will not be used but just amount to a waste of time. By conducting tests and involving end users in planning, the functions of a system will fix a problem and be of use (GitLab, no date).

Moreover, to organise the development work, I utilised a Kanban board with the stages of “To do”, “In progress”, “Testing” and “Done”. With each feature documented as a card, I was able to order them in priority and then transition from each stage as development progressed. Additionally, I noted dependencies where applicable to help prioritise work further. Finally, by applying due dates to each card, I stayed on-track for meeting each functional requirement I had documented for the system.

II. Development Stages

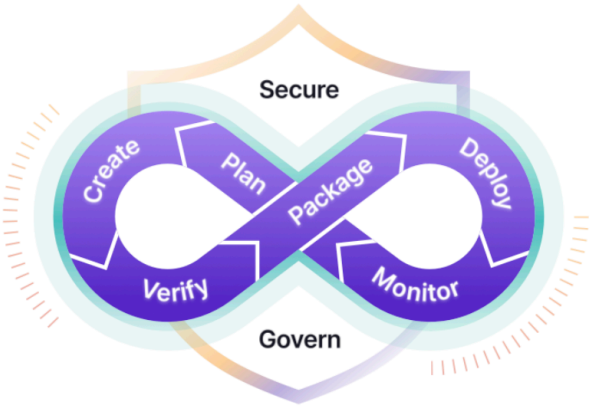

Figure 15. Stages of DevOps Software development depicted as a cycle starting from planing and ending at monitoring with the latter feeding back into the former (GitLab, no date).

Plan

I detailed a high-level plan of the project’s system within my proposal document shared with faculty at my University. This included an overview of the intended applications to be included and each feature that would reduce mean incident response time. Also, an infrastructure diagram was included to illustrate how components would interact.

Once I had discussed and finalised my project supervisor, I refined my project plan into a contract that laid out the business need, aims, objectives, deliverables, constraints and risks. This was all then reflected onto a Kanban board with estimated completion dates for a timeline of work.

Throughout the following stages, the approach taken to meet the objectives changed as better methods were identified. For example, I initially utilised bash scripts to automate the deployment of the development environment however as the project progressed this became unwieldy and therefore I migrated the deployment to ArgoCD which synchronised the Kubernetes cluster with code stored in version control (GitHub). This removed the complexity, allowed me to learn how ArgoCD worked and provided useful, live diagrams of the deployed infrastructure to help visualise the system (see Figure 21).

Create

Application code was written in Python due to its wide popularity, extensibility via 3rd party libraries and my personal knowledge being predominantly in the language. An object orientated approach was utilised via classes and objects to create a modular, reusable codebase which could be imported by other services where required.

Infrastructure code was written in Terraform for AWS resources and YAML for Kubernetes'. Terraform is a popular infrastructure-as-code language (Paz, 2023) with wide support for all major cloud service providers along with Kubernetes. The recommended method of defining Kubernetes resources via the official documentation is using YAML and therefore this approach was followed to ensure support and compatibility with 3rd party extensions used.

Source code was stored in GitHub as a central source of truth and connected to ArgoCD (further details outlined in the "Deploy" sub-section) via a GitHub app. Only Kubernetes resources were synced via ArgoCD as the AWS resources are idempotent but also can't be delivered via Argo.

Static data for the project such as secrets and configuration values were stored in AWS secrets manager and Kubernetes configmaps respectively. Secret values were defined at build to avoid storing them in version control which would contradict best practice (Ellingwood, 2017). The configmaps were attached to containers as files and read at runtime, inserted into the code to configure behaviour.

Verify

Code was tested within the Kubernetes cluster to ensure it would work effectively with other services. Throughout the development of application code, a priority was placed on producing useful log outputs to avoid ambiguous errors.

All services output logs to their containers standard output and can be read within the Kubernetes cluster. Not only is this recommended by the 12 Factor application model, it allows ArgoCD to collect said logs and make them available within its UI so command line access to the cluster is not required to troubleshoot errors.

Package

Once tests were completed and validated, application code and associated dependencies were packaged into container images and stored in Docker Hub.

Dependencies were installed via appropriate package managers (apt for system and pip for python libraries). Each service is ran as a non-root user for security purposes as a compromise of the container will have limited abilities to cause further damage.

Deploy

From Docker Hub, images are pulled and ran within Kubernetes pods. Associated configmaps are attached as files and secret values are associated to environment variables.

Custom Kubernetes pods and 3rd party extensions are deployed using ArgoCD. Helm charts are used to set variables within the Kubernetes manifest files and Kustomize is used to adapt the manifests to suit the environment.

Each application exposes a HTTP service via port binding to the cluster's internal network. Exceptions to this are the switchboard applications that are publicly exposed due to their need to receive data from Slack's API. To enhance their security, the switchboards will first authenticate the data received before forwarding it to the appropriate backing service by using a shared secret that both Slack and the switchboards have access too (see Figure 22).

Monitor

The state of running Kubernetes pods is monitored via the ArgoCD dashboard. This details the health, network traffic and sync status with source in GitHub. Log outputs can be read for each managed container and erroring pods can be destroyed and rebuilt using the self-heal functionality.

Any runtime errors experienced in the applications are outputted to standard output of the running container but also within slack to the user. This makes issues more visible and therefore can be quickly identified and remediated. Where problems are expected from user-error, advice on how to fix are provided.

Critical Analysis & Reflection

I. Successes

Firstly, the project was a great learning experience. I was able to cement my initial understanding of Containers, Kubernetes and AWS' cloud but also develop new knowledge within areas of continuous delivery via ArgoCD, a tool I had never used before. This learning allowed me to appreciate the power and flexibility provided by the configuration tools Kustomize and Helm for kubernetes resource manifests but also how to install it within a cluster and connect it to a GitHub repository. Additionally, the experience utilising ArgoCD for my dissertation project was of benefit to my employment at Matillion, working part-time as a cloud security engineer. I was able to lead my team's efforts of maintaining security tooling via ArgoCD, reducing our dependency on the infrastructure team for support.

Moreover, to efficiently deploy Knative via ArgoCD, I had to combine Helm and Kustomize which required me to develop a custom management plugin. The reason for this is because Knative doesn't have an officially supported Helm chart so I couldn't configure the installation using variables. Therefore, I had to create my own

Helm chart for the required custom resources and then utilise Kustomize to set the installation namespace and configure resources to support the other extensions running within the cluster. Out of all the development and innovation that stemmed from this project, this custom plugin was my most proudest. It allowed me to combine the power of both templating and customisation to Kubernetes resources in a seamless process via ArgoCD.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: helm-kustomize-plugin
  namespace: argocd

data:
  # Configure plugin
  plugin.yaml: |
    apiVersion: argoproj.io/v1alpha1
    kind: ConfigManagementPlugin
    metadata:
        name: helm-kustomize-plugin

    spec:
        version: v1.0

        init:
            command: [sh, /var/run/argocd/helm-kustomize-plugin/init.sh]

        generate:
            command: [sh, /var/run/argocd/helm-kustomize-plugin/generate.sh]

        discover:
            fileName: "./templates/kustomization.yaml"

        parameters:
          static:
            - name: helm-parameters
              title: "Helm Paramaters"
              tooltip: "Paramaters for values.yaml"
              required: true
              itemType: string
              collectionType: map


  # Download dependency chart(s)
  # Fromulate helm values from provided map, helm-paramaters, in app
  # Template manifests using values
  init.sh: |
    #!/bin/bash

    helm dependency build

    PARAMETERS=$(echo "$ARGOCD_APP_PARAMETERS" | /tools/jq -r '.[] | select(.name == "helm-parameters").map |
to_entries | map("\(.key)=\(.value)") | .[] | "--set=" + .')

    echo "ArgoCD Parameters: $ARGOCD_APP_PARAMETERS"
    echo "Paramaters: $PARAMETERS"

    echo ". $PARAMETERS" | xargs /tools/helm template --include-crds --output-dir rendered

  # Build templates with kustomize
  generate.sh: |
    #!/bin/bas

    /tools/kubectl kustomize "rendered/$(/tools/helm show chart . | /tools/yq ".name")/templates/"
```

Figure 16. Shows YAML code to define a Kubernetes Configmap for the helm-kustomize custom ArgoCD plugin. The plugin.yaml block defines a file which sets the configuration for the plugin such as which commands to run and the prerequisites required for the plugin to be selected. The init.sh yaml block includes the code to both create the templates from the Helm chart and then apply said templates via Kustomize.

Before I began any development work, I defined a list of tasks I wanted the system to automate for incident response. These were inspired by my experience of containing security incidents during my placement and therefore reflected the response of real-world compromises. The final system automated 7 actions, all with a focus on the most common AWS resources to be compromised (Piper, 2023; Baguelin, Dereeper and McCloskey, 2023), Elastic Cloud Compute (EC2) and Identity and Access Management (IAM). The system efficiently automates these actions, removing the need for incident responders to repeat tasks but doesn't remove them from the equation. This collaboration instills trust and reliability into the automation as it won't blindly respond to any detections but has to be prompted beforehand, allowing for analysis of the situation first. Furthermore, restrictions can be defined to require approval when committing an action against specific resources or as a whole. This allows security teams to work with their infrastructure counterparts, maintaining the security of their cloud estate but also reduce the impact on production systems and potentially customer facing outages.

I wanted to make my system as user friendly as possible, providing methods of input that people from different technical backgrounds could understand and utilise. Because of this, I decided to facilitate natural language prompts by users, alongside the traditional declarative input method. Utilising OpenAI's Large Language Model (LLM), ChatGPT, I was able to determine the action and targets from user's inputs that didn't follow a specific format. I experimented with multiple system prompts to inform the LLM on its role and instructions on how to respond, with the best outputs resulting from providing both descriptions of the supported actions and example correct responses from prompts. By providing tuning data, ChatGPT's responses were both reliable and made the system more easy to use, so much so that if I were to develop the system again, I would heavily consider only supporting natural language input.

```
You are going to be used to extract specific pieces of information from prompts written in natural English
language that aim to complete security incident response actions within an AWS account. It is your job to
determine the targets of the action and what action the user wants to take from the following list. Descriptions
of each action will be supplied to help improve your understanding of what they do. If you can't determine the
action from the prompt, or an action is not directly tailored to the prompt, please respond with “Action:
unsupported”. If you can't determine the targets of the action, please respond with “Targets: unknown”. For the
targets you identify, please respond with them in a list, for example “Targets: ”target 1”, “target 2”, …”. Do
not include the target's resource type in your response, only its name. Do not respond with the target AWS
account ID.

Here is the list of supported actions with their description.
- help: Lists all possible commands
- stop-ec2: Stops EC2 instance
- network-contain-ec2: Applies deny all security group to EC2 instance
- snapshot-ec2: Create snapshot of all volumes attached to EC2 instance
- iam-contain-ec2: Applies deny all policy to EC2 instance's instance profile
- refresh-user-session: Applies policy to IAM user to invalidate temporary credentials made
- refresh-role-session: Applies policy to IAM role to invalidate temporary credentials made
- disable-user: Applies deny all policy to IAM user
- disable-role: Applies deny all policy to IAM role

Here is a list of example prompts and their correct responses that you should base your answers on.
- prompt: Block network access to i-xxx and i-yyy in us-east-1 00000000000
- response: {'Targets': ['i-xxx', 'i-yyy'], 'Action': 'network-contain-ec2'}

- prompt: Prevent i-xxx in 00000000000 and eu-west-1 from accessing other AWS resource
- response: {'Targets': ['i-xxx'], 'Action': 'iam-contain-ec2'}

- prompt: Take a copy of i-zzz in 00000000000 us-east-1
- response: {'Targets': ['i-zzz'], 'Action': 'snapshot-ec2'}

- prompt: Stop i-zzz in 00000000000 us-east-1
- response: {'Targets': ['i-zzz'], 'Action': 'stop-ec2'}

- prompt: Disable jarvis-demo user in 00000000000
- response: {'Targets': ['jarvis-demo'], 'Action': 'disable-user'}

- prompt: Remove jarvis-demo role's permissions in 00000000000
- response: {'Targets': ['jarvis-demo'], 'Action': 'disable-role'}

- prompt: Invalidate any STS tokens created from jarvis-demo role in 00000000000
- response: {'Targets': ['jarvis-demo'], 'Action': 'refresh-role-session'}
```

Figure 17. Shows an example system prompt sent to ChatGPT. The first paragraph is static, this doesn’t change. The list of supported actions and the list of prompt examples are dynamically populated from Kubernetes Configmaps so as new actions are added, ChatGPT can support associated natural language prompts.

II. Improvements

Firstly, at the beginning of development, I deployed all the cloud and Kubernetes resources via bash scripts. These files contained all the commands needed to create the resources and configure them. As my progress advanced, maintaining these files became untenable due to their vast size and increasing complexity. Therefore, I migrated the process to ArgoCD which synced all the code that defined the resources to be built and their configuration stored in GitHub to the deployment environment. In hindsight, this should have been my approach from the start as I had to re-work a lot of the process defined in the bash scripts for ArgoCD, resulting in unnecessary work which could have been better focused on developing and testing new features.

To improve a user’s experience of the system I built for this project, specific documentation on usage and explaining its functionality would had been of benefit. I contributed limited effort to this by facilitating a help command which outputted a short description and usage directions for each command. However, a more detailed version, most likely outside of the application and stored alongside the relevant code, would increase the longevity of the developed system and allow others to utilise the tool completely without my input and guidance.

Furthermore, I continuously ensured that each service I programmed outputted useful logs so any issues could be troubleshooted. Meanwhile, upon reflection it would had been more efficient to create a standardised logging library which would be imported and used by each service instead of writing the log-cases individually. The benefit of this improvement would implement a general format across all the logs and create a central point from which logs could be stored in long-term storage if needed. This functionality would not have drawn considerably development time to achieve however did not contribute to the project's core objective and therefore wasn't prioritised.

Additionally, I explained above that the integration with OpenAI's ChatGPT LLM was successful, producing reliable results from my test prompts. However, given more time to develop this feature, I would have incorporated a human validation step between the LLM's response and enacting the action it deemed appropriate. By asking the user to approve of the generated answer, it would improve the reliability and safety of using the Artificial Intelligence (AI) as an incorrect answer could be rejected, either retrying the prompt or using the declarative command method instead. Moreover, I could use the user's input to further train the model on my use case. By providing feedback to the model on what good responses to prompts are, this would improve its accuracy in picking the correct action corresponding to the user's request. Likewise, the model would be able to learn from its mistakes where it chose the wrong option.

III. Appraisal Of Final Product

From the outset, I defined clear functional objectives for my project in the form of features that a user could utilise to reduce their incident response time for security incidents within a cloud environment. I am proud to say that the system developed meets all of these objectives well in a modular, efficient manner that can be easily built upon in the future.

The entire infrastructure on which the system runs is built via code. This allows for efficient maintenance of resources, reduces errors from misclicks and can be easily used to replicate the system I built myself. I used Terraform due to its wide support of all the different services I used (AWS, Kubernetes, TLS certificates) and YAML to define the Kubernetes resources, organised into Helm charts and deployed using ArgoCD. I had experience writing and deploying infrastructure with Terraform however my knowledge of ArgoCD was very limited so learning this tool was of great use and appreciation.

All response actions I wished to implement were completed, along with the ability to control who can change resources within the cloud environment through an approval process. This functionality was inspired from my time working in industry as a cloud security engineer for placement, during which we would liaise with asset owners, especially those running customer-facing resources, when responding to security incidents. It is paramount that security teams not only ensure security of infrastructure but do so in the least destructive manner, often-which requires guidance from the engineers and developers who built the system.

The implementation of ChatGPT to process natural language prompts was of great success and incorporates a bleeding-edge technology into my project. Whilst implementing this, I was able to develop my understanding and experience of prompt engineering to improve the reliability of the AI's responses. Additionally, the system prompt provided to the LLM is only an estimated 750 tokens, along with the users prompt which is likely to be less than 100, means that the cost of using OpenAI's API access is negligible. During testing, I only spent $0.01 for sending and receiving a total of 21 726 tokens.

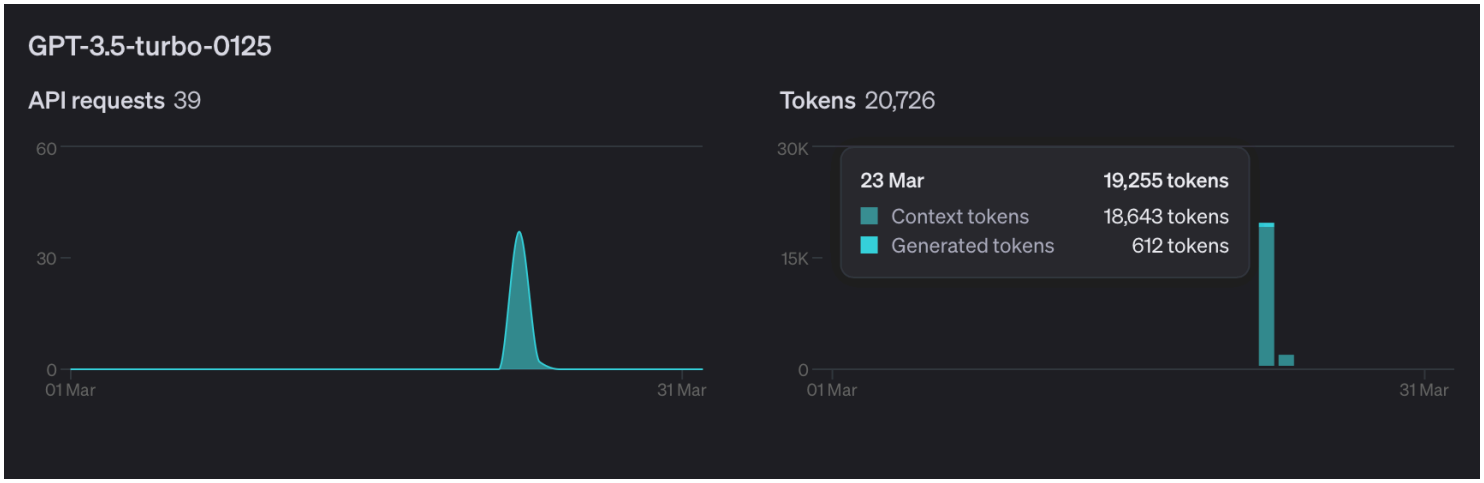

Figure 18. Graph of ChatGPT spend from usage during development and testing.

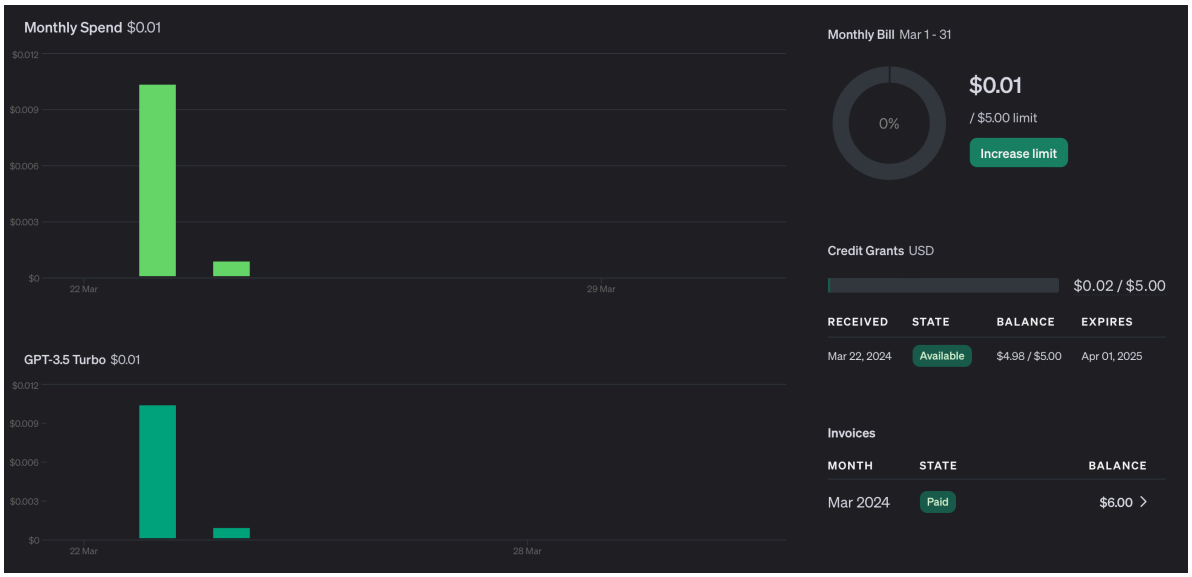

Figure 19. Graphs of requests amounts and associated cumulative token count accrued during development and testing.

Figure 20. Responder user submits a natural language prompt to prevent the "jarvis-demo-no-approval" role from being able to commit actions in the AWS account 637423250821. The automation has responded with a success message that a deny-all policy has been applied to the role. This conversation was facilitated by ChatGPT through the LLM determining what the desired action was and what the target(s) were.

My ultimate goal for developing this automation was to reduce the time it took to contain a security incident. After measuring both myself taking the actions within AWS' Management Console and the automation's I was able to empirically demonstrate the system meeting this goal, as shown in the following table. It is also worth noting that as the number of targets an action / actions need to be committed against, the human response time will increase too. In the times shown below, there was only 1 targeted resource.

Critical Analysis & Reflection - Appraisal of Final Product - Incident Response Times

		Human Response Time (Seconds)		Automation's Response Time (Seconds)	
	Action	1 Target	2 Targets	1 Target	2 Targets
1	Disable IAM Role	20	43	22	29
2	Disable IAM User	19	41	23	29
3	Refresh IAM role's temporary credentials	16	34	21	27
4	Refresh IAM user's temporary credentials	23	43	23	28
5	Contain EC2 instance's IAM capabilities	19	45	25	35
6	Contain EC2 instance's network abilities	15	37	19	25
7	Take snapshot of EC2 instance	21	40	20	27
8	Stop EC2 instance	12	19	18	24
9	Contain EC2 instance by conducting actions 5 - 9	57	118	58	93
	Average time difference	3	-13	-3	13

To calculate these results, I started timing myself from the homepage of AWS' Management Console and from first typing the command in Slack. This gives an accurate response time that not only measures the automation's speed on conducting the actions but also the required human responder's input. I didn't include response times using natural language prompts because the length of the input to the system is variable depending on how concise the incident responder is whereas the declarative commands are the same each time. If the developed system automatically triggered from a detection alert, without relying on an incident responder to direct it, these times would be drastically lower for either target amount types.

From this experiment, we can observe that when responding to an incident concerning a single target, and only a single action is taken, manual intervention is faster. However, as both the number of targets grow along with or separately the number of actions, it is more efficient to utilise automation due to the reduced user interface clicks required.

Furthermore, whilst conducting this experiment, I quickly appreciated the ease of response compared to traditional manual effort. When using the management console, I had to navigate through multiple pages to complete actions and also remember the steps for each. Whereas, I only needed to send one message in the slack channel and the automation conducted them all for me seamlessly.

Overall, I focused my efforts throughout the development of my project to produce a reliable, error-free application that uses industry standard methods. However, there are multiple areas I see for improvement that would increase the value of my system. For example, a validation step for ChatGPT's responses would guard against any unwanted actions being committed but also allow for dynamic training of the model, learning from its successes and failures while a user accepts or rejects its suggestions. Additionally, other incident response actions that focused on more complex cloud resources would demonstrate my system's ability to respond to all manner of compromises.

IV. Analysis Of Approach

The techniques and procedures I used to develop the system were all inspired by my time working at Matillion during my placement and further as a part time security engineer. I gained invaluable experience not only in how incident response works within an enterprise environment but also through working alongside infrastructure and software engineers, I witnessed and supported the development of a product that was used by large corporations to extract, transform and load their sensitive data.

Firstly, I defined all my resources in code to which could be used to build and manage them whenever was needed. There was no need to document the steps to create and configure services within the AWS account through using the Management Console because the infrastructure-as-code outlines what is used and how it is setup. This approach worked well and scaled efficiently as the system grew. I could create multiple instances of the same resource easily without having to duplicate efforts and could efficiently manage costs by only running the system when needed.

Likewise, for workloads running in Kubernetes, I defined them using the YAML data serialisation language. These were then organised into Helm charts for templating and stored in a private GitHub repository. YAML is the default method of defining Kubernetes resources and so this was the only choice however the use of Helm was optional. I chose to incorporate the charts instead of just plain YAML manifests because they offer the ability to configure the installation in-code without the need to change resources when they are running. This was especially efficient when deploying open-source, 3rd party applications to the Kubernetes cluster to extend its functionality. These worked seamlessly with ArgoCD, maintaining synchronisation between the runtime cluster environment and the intended state defined in code.

ArgoCD is a continuous delivery application for Kubernetes resources. It works by reading code from a source-code repository, like GitHub, and applies said code to the defined destination. It responds to changes in the cluster to ensure the runtime state always reflects the desired code and removes the need for admin access to the cluster. Furthermore, ArgoCD provides a graphical representation of all the resources currently running in the Kubernetes cluster that it manages which helps illustrate how components collaborate to form the overall system. The decision to use Argo was defined as a risk in my project contract because I had no practical experience with it. Therefore, I did not begin using it until later stages of development and instead favoured deploying the Kubernetes resources manually via admin cluster access. In hindsight, this delay was a mistake as it took considerable time to migrate the resource manifests from my local deployment method to work with ArgoCD. However, the use of the tool was of great benefit, the predominant reason for the migration was due to the difficulty of managing all the resources myself and once it was completed, I was able to develop the system more quickly as the platform maintained all the resources for me.

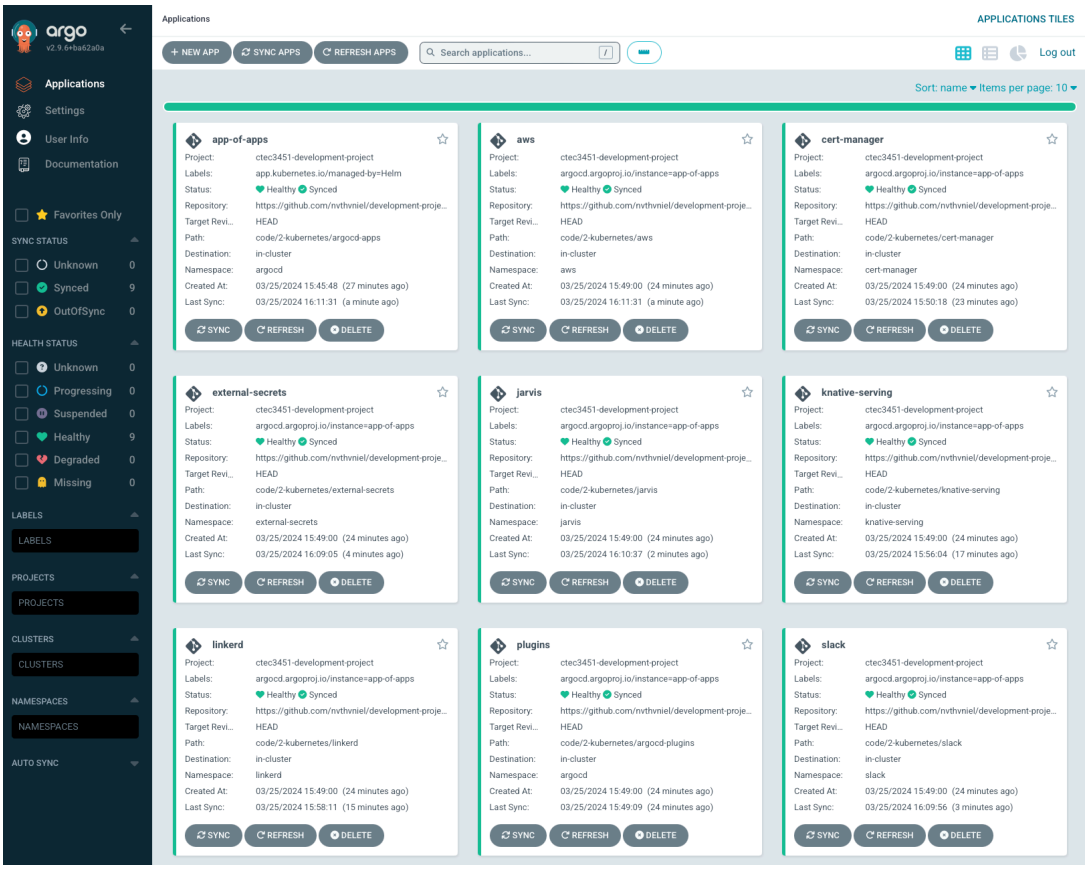

Figure 21. View of ArgoCD applications installed within cluster with their health and sync status

I used open-source applications to provide extended functionality in the Kubernetes cluster. Instead of re-inventing the wheel, using these allowed me to focus my efforts on developing the core system that would reduce incident response times. All of these extensions proved useful and reliable except one that I found out after implementing interactive responses in Slack to be inadequate. I originally planned to utilise Triggermesh as a gateway to accept and authenticate incoming requests from Slack's API. This worked for events that triggered from users sending messages however when they click on interactive buttons to approve actions, I realised Triggermesh did not support these types of events. Therefore, as the validation Slack process was a core component to my system, I decided to replace Triggermesh with my own custom code. Using Slack's documentation (Slack API, no date), I was able to successfully authenticate traffic sent from Slack and then route requests from users to their appropriate services. Consequentially, this approach proved an improvement over Triggermesh as, due to its simplicity, the latency between a user's message / interaction in Slack to a response being produced by the appropriate service noticeably reduced. Additionally, I was able to pass prompts that didn't match any defined routes to ChatGPT for clarification on the user's desired action, a process that I envision would have been considerably more complex if using Triggermesh.

```
...
def auth_event(self, slack_timestamp, slack_signature, body):
        logging.info(f"\n[+] Authenticating Event ...")
        authed = False

        local_timestamp = int(time.time())

        logging.info(f" | Slack timestamp: {slack_timestamp}")
        logging.info(f" | Local timestamp: {local_timestamp}")

        # Event is > 5 minutes old
        if ( (local_timestamp - slack_timestamp) > 60 * 5 ):
            logging.info(f" | Possible replay attack, skipping event ...")

        else:
            signing_secret = os.getenv("SLACK_SIGNING_SECRET")

            logging.info(f" | Slack Signing Secret: {signing_secret[0:5]}...")

            signing_secret = bytes(signing_secret, "utf-8")

            version_number = "v0"
            sig_basestring = f"{version_number}:{slack_timestamp}:{body}"

            sig_basestring = bytes(sig_basestring, "utf-8")

            basestring_hash = hmac.new(
                signing_secret,
                sig_basestring,
                hashlib.sha256
            ).hexdigest()

            logging.info(f" | Slack signature: {slack_signature}")

            local_signature = f"v0={basestring_hash}"
            logging.info(f" | Local signature: {local_signature}")

            if ( hmac.compare_digest(local_signature, slack_signature) ):
                authed = True
                logging.info(f" | Signatures match, authentic")

            else:
                logging.info(f" | Signatures differ, not authentic")

        return authed
...
```

Figure 22. Python code within events-switchboard and interactions-switchboard's server method to authenticate received data to ensure it originated from slack.

For my custom code to implement incident response actions and handle user's incoming requests, I used Python. This programming language is widely popular and I myself had extensive experience using it. Its simplistic syntax yet wide array of functionality proved a great asset in this project. I was able to efficiently and quickly develop prototypes in which could be iteratively developed upon to enrich a user's experience and Improve functionality. I have experience in other programming languages such as PHP and C#, both of which, in my opinion, would have taken considerably longer to use for this project due to their more complex syntax.

Overall, my approach produced a sound and capable system that reflected leading industry practices. I learnt considerable amounts of new knowledge during the development along with compounding my existing experience too. The final product achieve the project's goal of reducing incident response times by containing resources via a user friendly interface.

References

1. AWS. (No date - a) "Amazon DynamoDB". Available at: https://aws.amazon.com/dynamodb/ (Accessed: 02/12/2023).

2. AWS. (No date - b) "Time to Live (TTL)". Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/TTL.html (Accessed: 03/12/2023).

3. AWS. (No date - c) "IAM roles for service accounts". Available at: https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html (Accessed: 26/03/2024).

4. AWS. (No date - d) "Pricing for On-Demand Capacity", Amazon DynamoDB. Available at: https://aws.amazon.com/dynamodb/pricing/on-demand/ (Accessed: 26/03/2024).

5. Atlassian. (no date) "Incident management for high-velocity teams". Available at: https://www.atlassian.com/incident-management/on-call/alert-fatigue#What-is-alert-fatigue? (Accessed: 25/03/2024).

6. Baguelin, F., Dereeper, C.T. and McCloskey, M. (2023) "Following attackers' (cloud)trail in AWS: Methodology and findings in the wild", Datadog Security Labs. Available at: https://securitylabs.datadoghq.com/articles/following-attackers-trail-in-aws-methodology-findings-in-the-wild/ (Accessed: 26/03/2024).

7. Check Point. (no date) "Cloud Incident Response". Available at: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-incident-response/cloud-incident-response/ (Accessed: 25/03/2024).

8. Cloud Control. (no date) "A guide to Kubernetes and Cloud migration to cut costs and maximise Benefits". Available at: https://www.ecloudcontrol.com/a-guide-to-kubernetes-and-cloud-migration-to-cut-costs-and-maximize-benefits/ (Accessed: 26/03/2024).

9. GitHub. (no date) "About secret scanning". Available at: https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning (accessed: 27/04/2024).

10. Jackson, M. (2023) "Best Practices for Managing and Storing Secrets Including API Keys and Other Credentials [cheat sheet included]". Available at: https://blog.gitguardian.com/secrets-api-management/ (Accessed: 03/12/2023).

11. Jaworski, P. (no date) "Why companies are switching to Kubernetes". Available at: https://stackshare.io/posts/companies-using-kubernetes-in-production-2018 (Accessed: 26/03/2024).

12. Knative. (no date - a) "Install the networking layer". Available at: https://knative.dev/docs/install/operator/knative-with-operators/#install-the-networking-layer (Accessed: 26/03/2024).

13. Knative. (no date - b) "Installing cert-manager for TLS certificates". Available at: https://knative.dev/docs/install/installing-cert-manager/ (Accessed: 26/03/2024).

14. kubernetes.com. (2023) "Kubernetes Components". Available at: https://kubernetes.io/docs/concepts/overview/components/ (Accessed: 02/12/2023).

15. Let's Encrypt. (no date) "About Let's Encrypt". Available at: https://letsencrypt.org/about/ (Accessed: 26/03/2024).
16. Linkerd. (no date) "Overview". Available at: https://linkerd.io/2.15/overview/ (Accessed: 26/03/2024).

17. McCune, R. (2023) "Container security fundamentals: Exploring containers as processes", Datadog Security Labs. Available at: https://securitylabs.datadoghq.com/articles/container-security-fundamentals-part-1/#containers-are-just-processes (Accessed: 26/03/2024).

18. Morgan, W. (no date) "What every software engineer needs to know about the world's most over-hyped technology", Buoyant - Service Mesh Manifesto. Available at: https://buoyant.io/service-mesh-manifesto (Accessed: 02/12/2023).

19. Piper, S. (2023) "How to get rid of AWS access keys - Part 1: The easy wins", Wiz Blog. Available at: https://www.wiz.io/blog/how-to-get-rid-of-aws-access-keys-part-1-the-easy-wins (Accessed: 26/03/2024).

20. Scarlett, R. (2023) "Why Python keeps growing, explained". Available at: https://github.blog/2023-03-02-why-python-keeps-growing-explained/ (Accessed: 26/03/2024).

21. Slack API. (no date) "Verifying requests from Slack". Available at: https://api.slack.com/authentication/verifying-requests-from-slack (Accessed: 25/03/2024).

22. Sparxsystems.com (no date) "UML 2 Tutorial - Use Case Diagram". Available at: https://sparxsystems.com/resources/tutorials/uml2/use-case-diagram.html (Accessed: 09/11/2023).

23. Wiggins, A. (2017) "The Twelve-Factor App". Available at: https://12factor.net/ (Accessed: 01/12/2023).

Appendices

Appendix A - Slack Bot Configuration

Note — sensitive data points have been censored in grey as these are not to be shared publicly

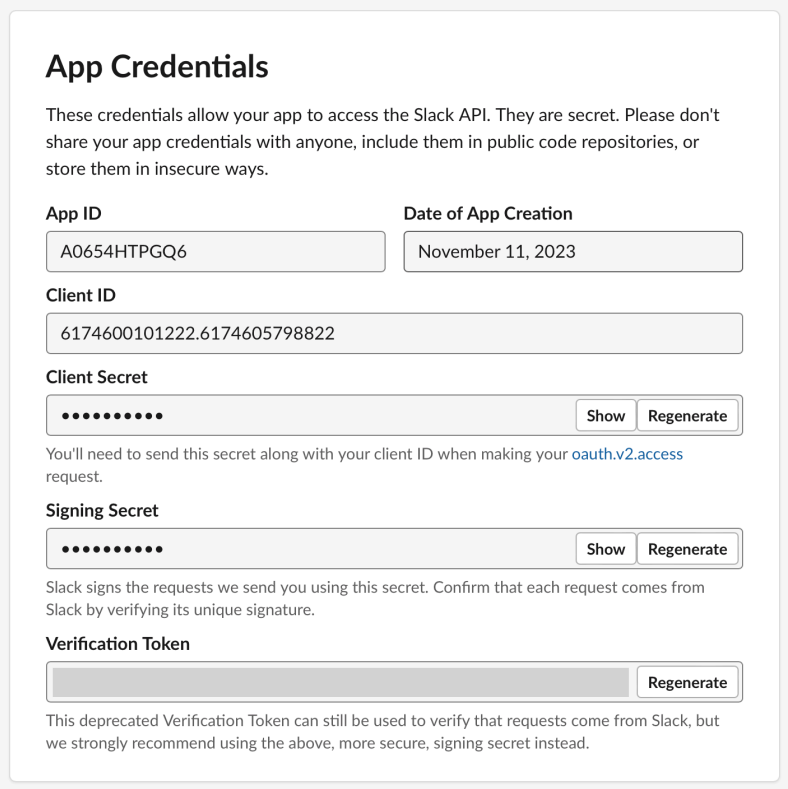

Figure 23. The App Credentials found in the bot's Basic Information page. The Signing Secret is used to authenticate received messages from Slack to ensure a malicious actor isn't trying to forge requests.

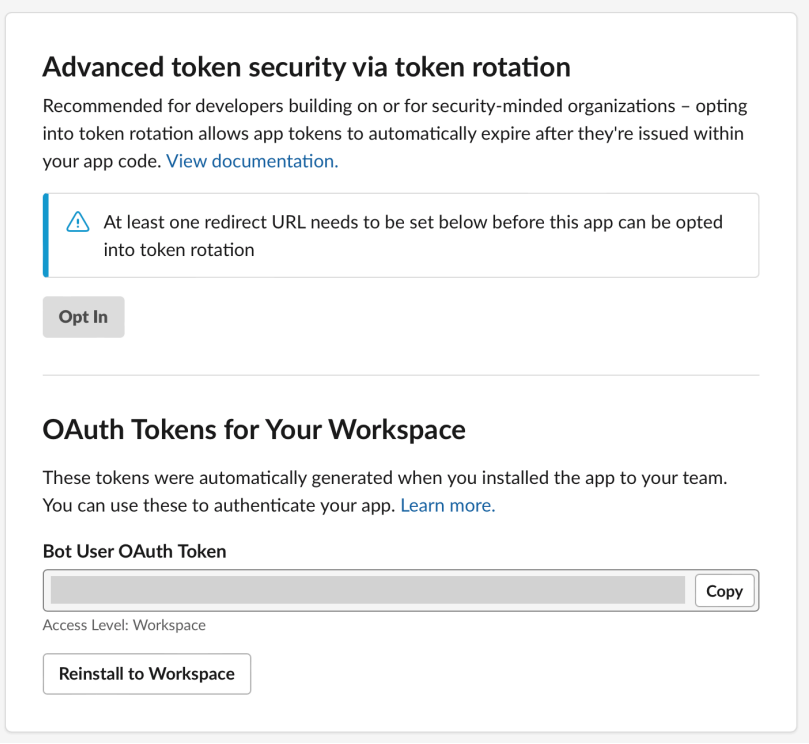

Figure 24. The OAuth token located within the bot's OAuth & Permissions page is used to authenticate to Slack's API so user data can be retrieved and messages sent in the Slack channel.

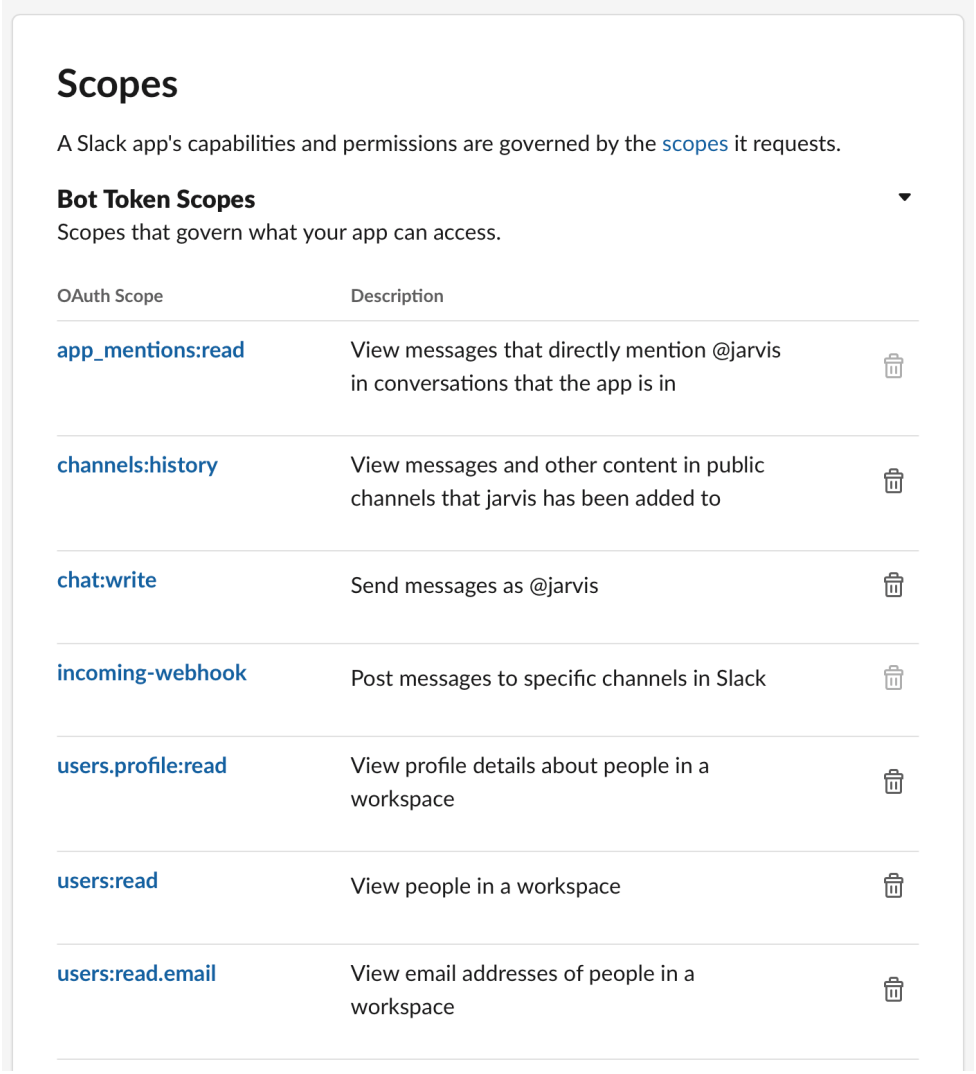

Figure 25. Additionally in the OAuth & Permissions page, the bot's permissions are set. These define what the token in Figure 14 is authorised to do via Slack's API.

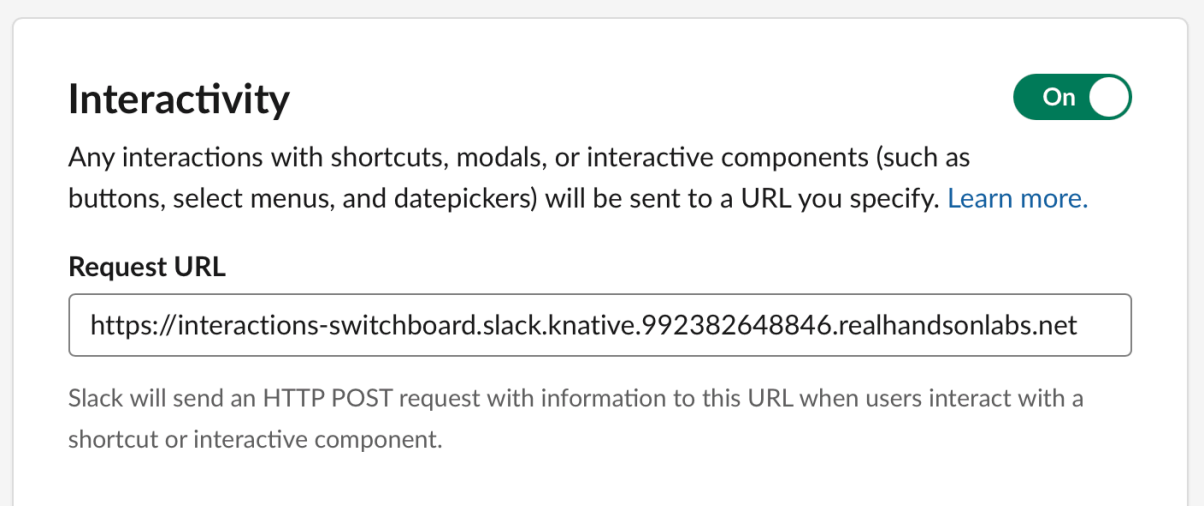

Figure 26. Configuring an endpoint to send data when a button is pressed in Slack is defined within the Interactivity & Shortcuts Page.

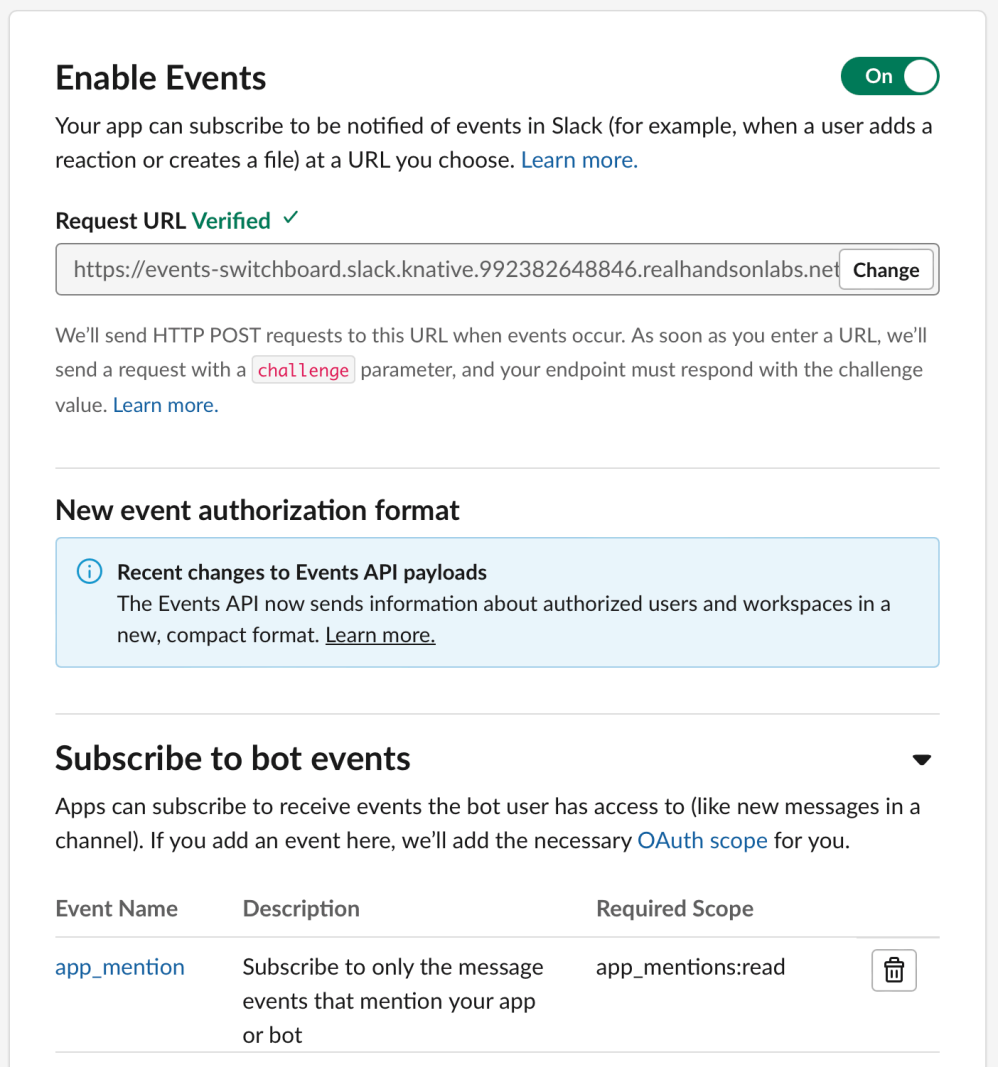

Figure 27. Within the Event Subscription page, the URL to receive events (slack messages posted by users in a channel) is defined. This has to be verified by responding to the challenge message sent by slack. Moreover, The bot is subscribed to "app_mention" events; it will only be sent an event from Slack when a user mentioned the bot using "@jarvis".

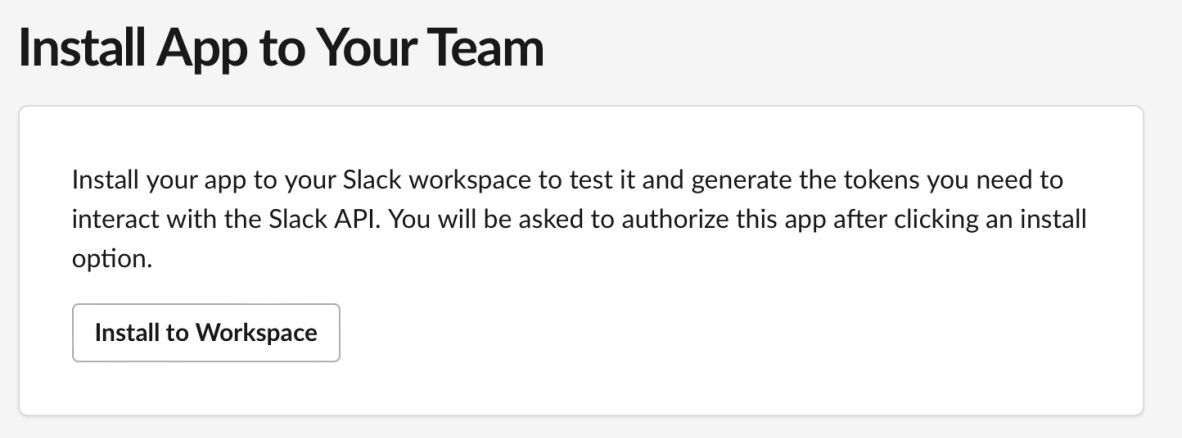

Figure 28. Once you have finished configuring the bot, you will need to install it to your Slack Workspace.

Appendix B - Creating Slack Channel & Installing Bot

- It's recommended to create a dedicated Slack channel to interact with the Bot

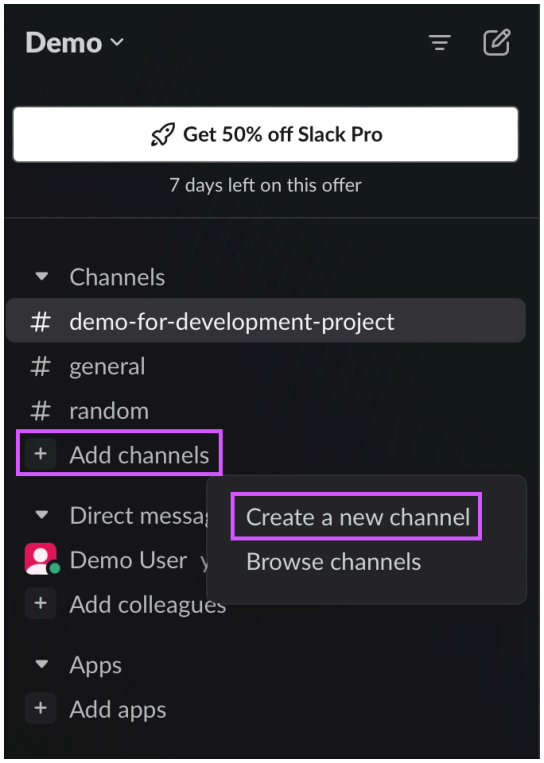

Figure 29. Click "Add channels" then "Create a new channel".

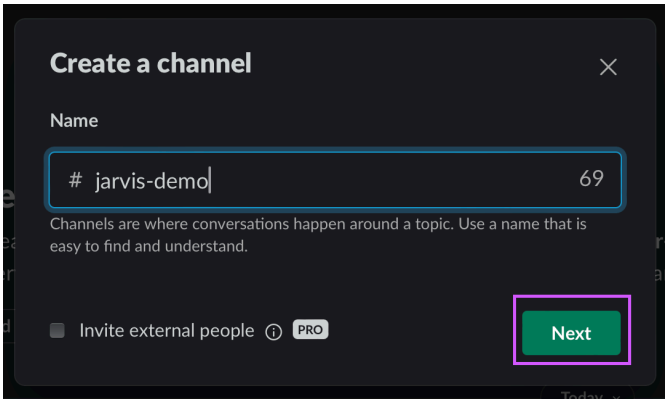

Figure 30. Enter a unique name for the channel then click "Next".

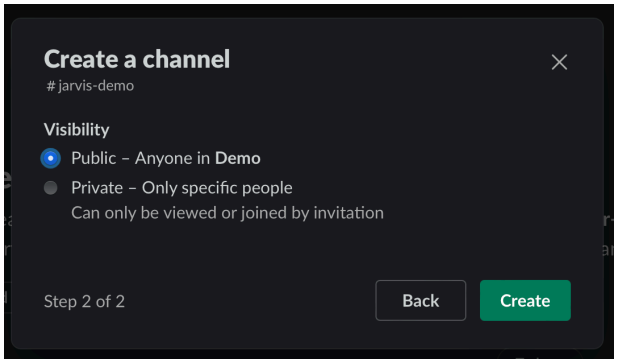

Figure 31. Configure the access level of the channel then click "Create"

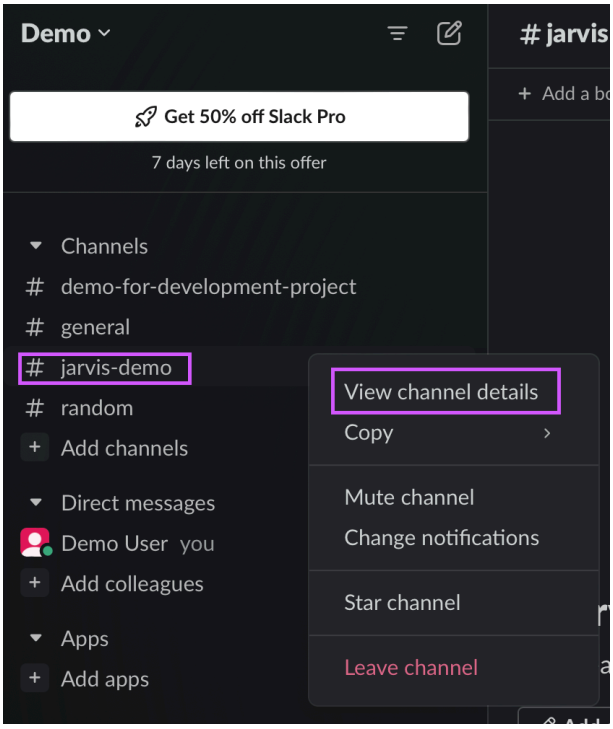

Figure 32. Right click on the channel and then open the "View channel details" tab.

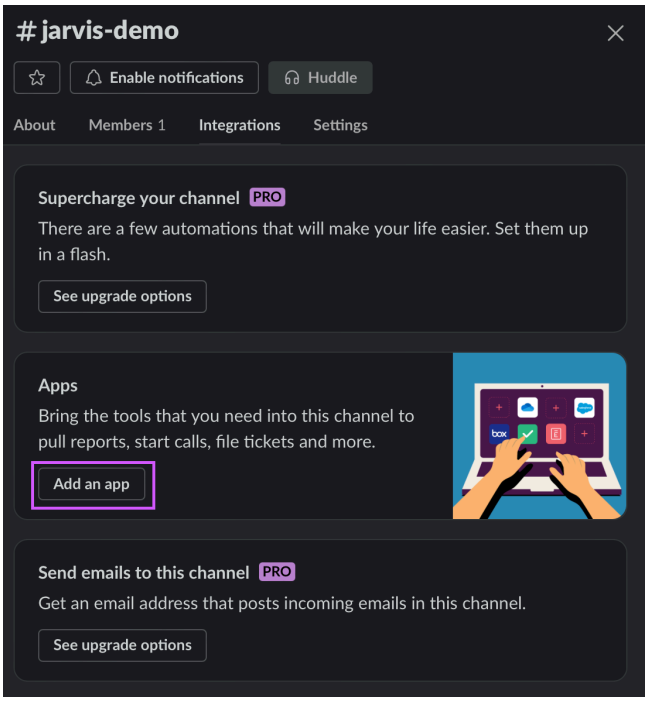

Figure 33. Navigate to the "Integrations" tab and click on "Add an app"

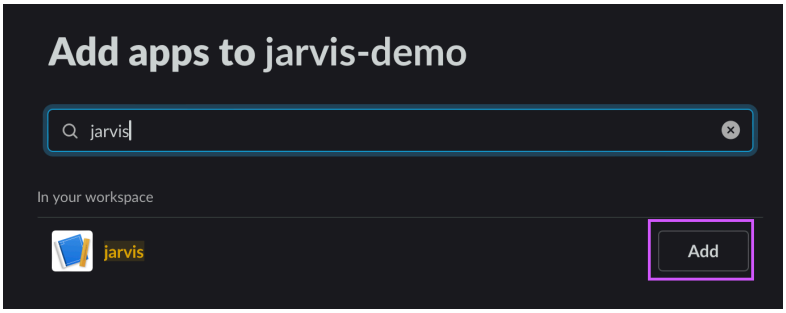

Figure 34. Search for the app by name and then click "Add".

Appendix C - Creating GitHub App

- ArgoCD is used to install and manage the Kubernetes resources
- The code for said resources should be stored in source control, in this case I have used GitHub
- For ArgoCD to read the resources' code in GitHub, it requires credentials to authenticate. The GitHub app facilitates this requirement
- ArgoCD's documentation on repository connection can be found here: https://argo-cd.readthedocs.io/en/stable/user-guide/private-repositories/

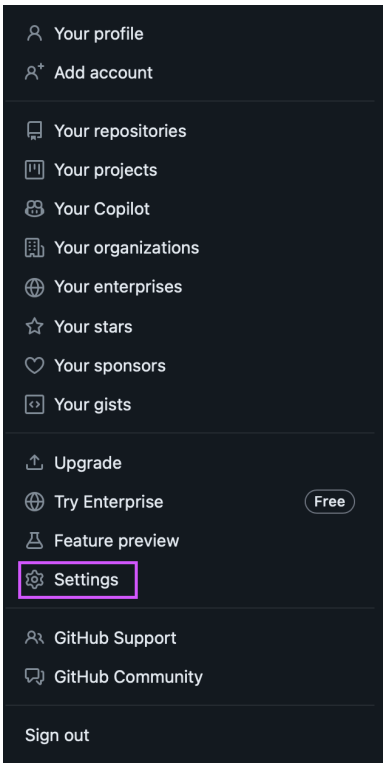

Figure 35. Open the "Settings" tab within your GitHub account.

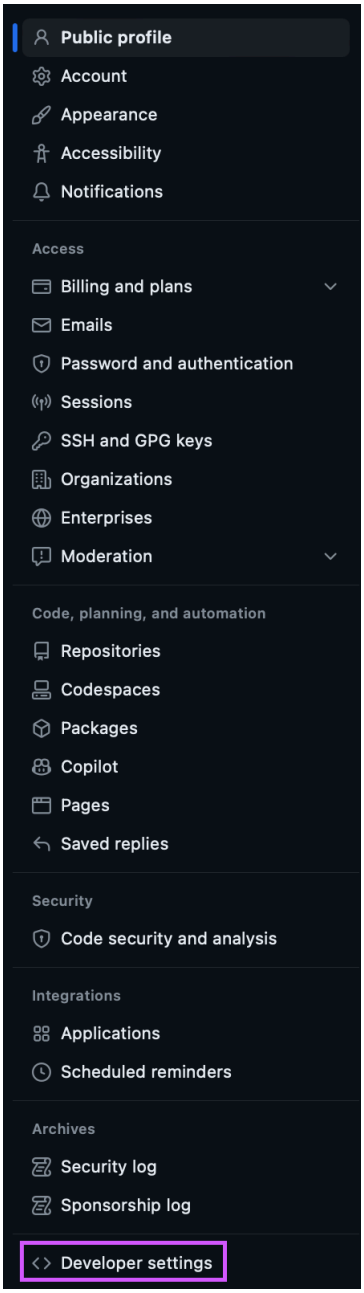

Figure 36. Open the “Developer setting” tab.

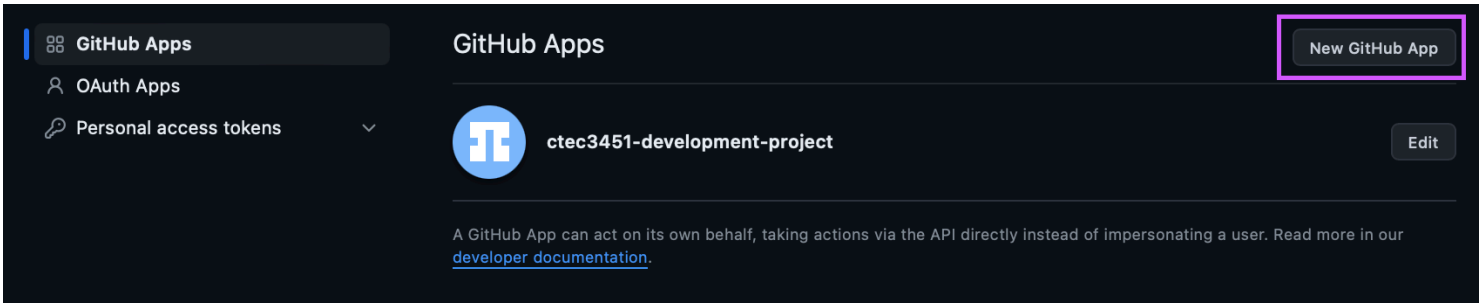

Figure 37. Within the “GitHub Apps” tab, click on “New GitHub App”.

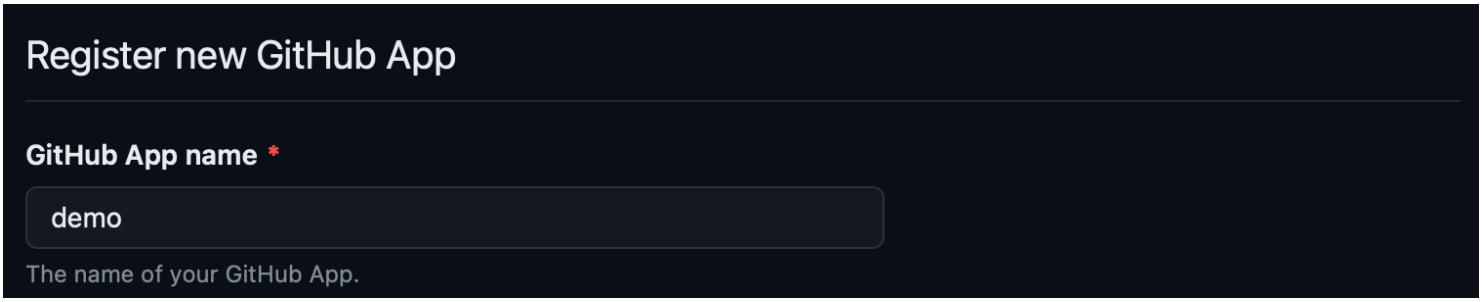

Figure 38. Give your app a unique name.

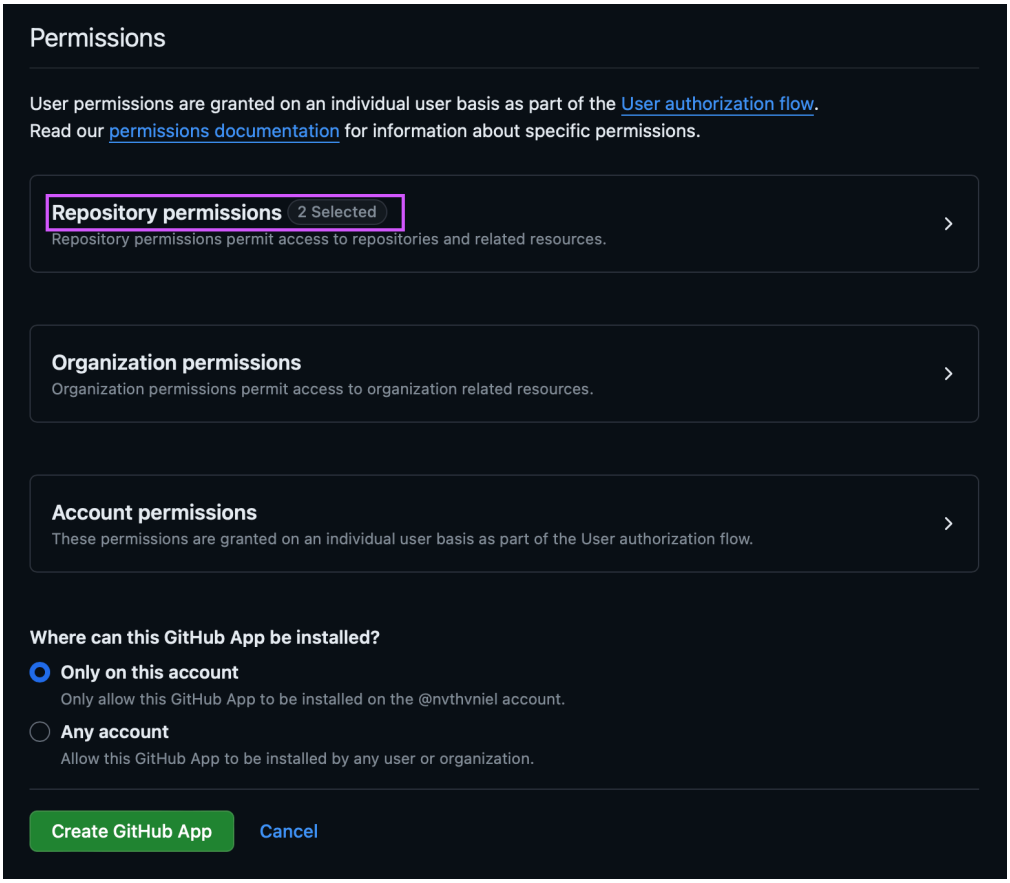

Figure 39. Enable read-only repository permissions for Contents and Metadata then click “Create GitHub App”.

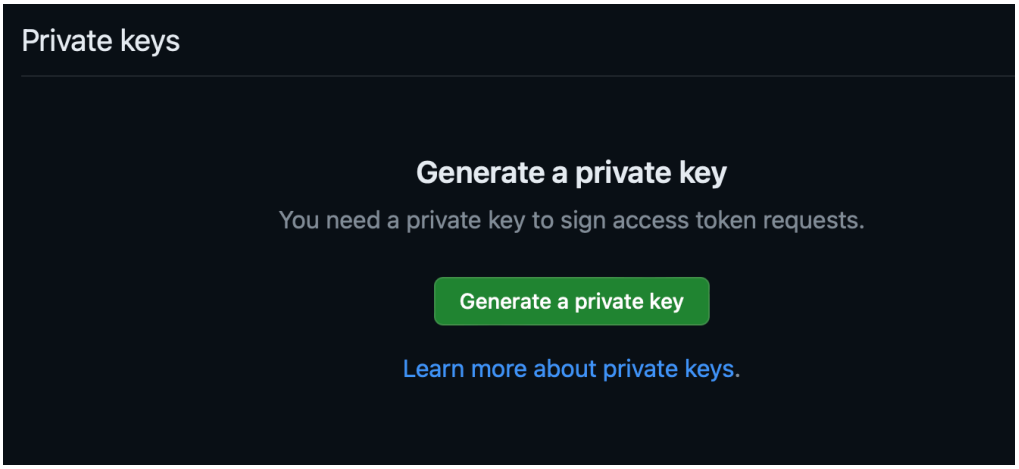

Figure 40. Within the “General” tab, scroll down to Generate a private key pair. This will download a file which will contain your private key, allowing ArgoCD to authenticate and read the code within GitHub.

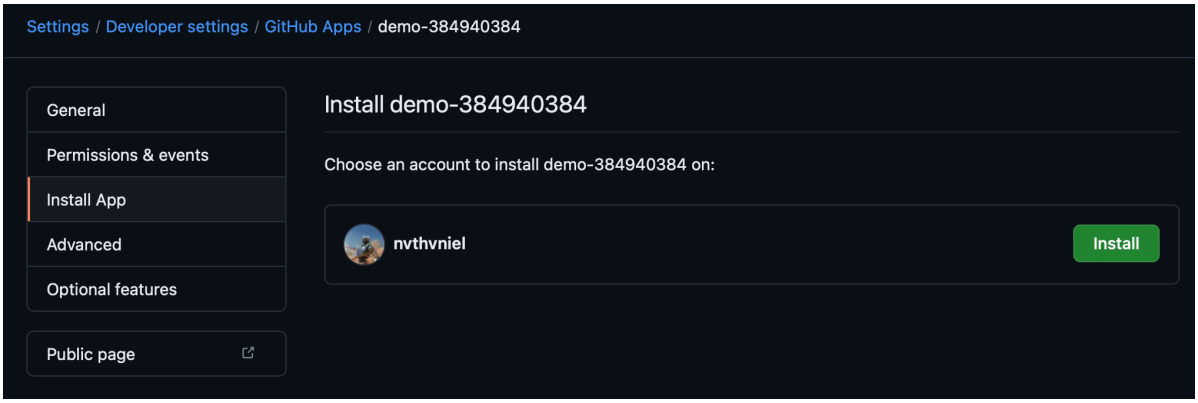

Figure 41. Open the “Install App” tab and click “Install”

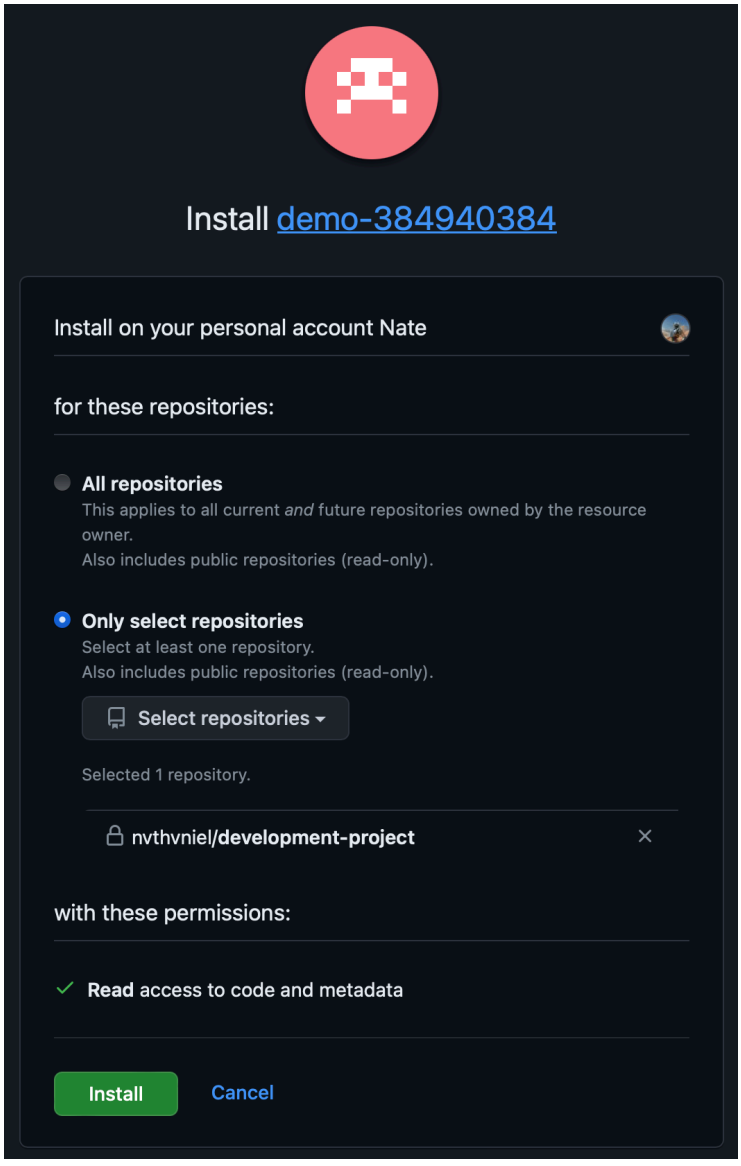

Figure 42. Select a repository that the app will have the permissions defined in Figure 39 then click “Install”.

Appendix D - Creating ChatGPT API Token

- ChatGPT API Reference Documentation: https://platform.openai.com/docs/overview

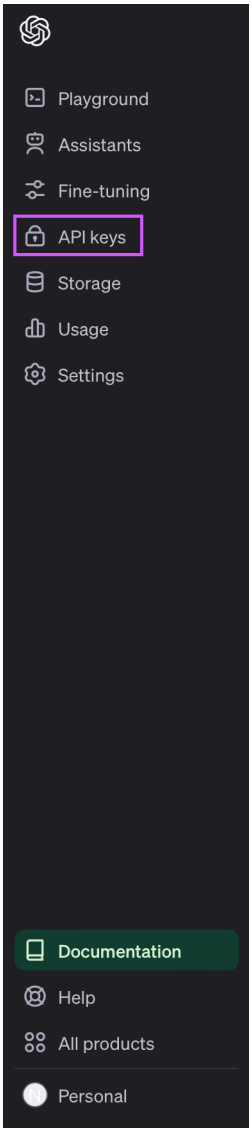

Figure 43. Open the API keys tab from the sidebar of your ChatGPT account.

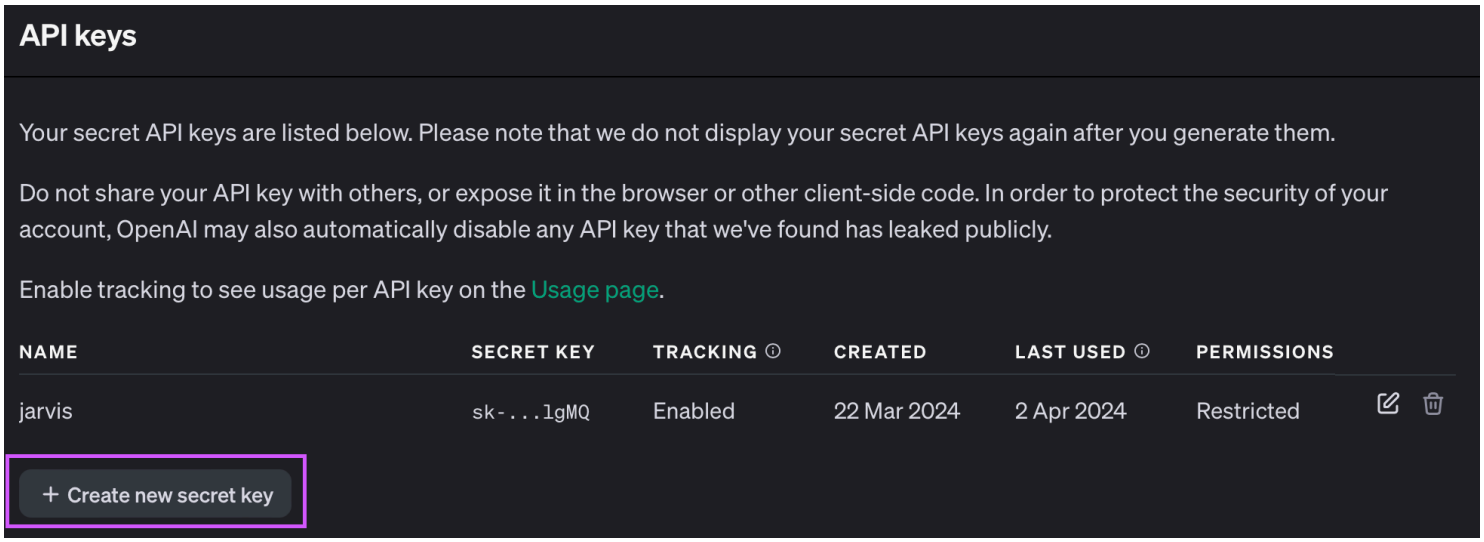

Figure 44. Click on “Create new secret key”

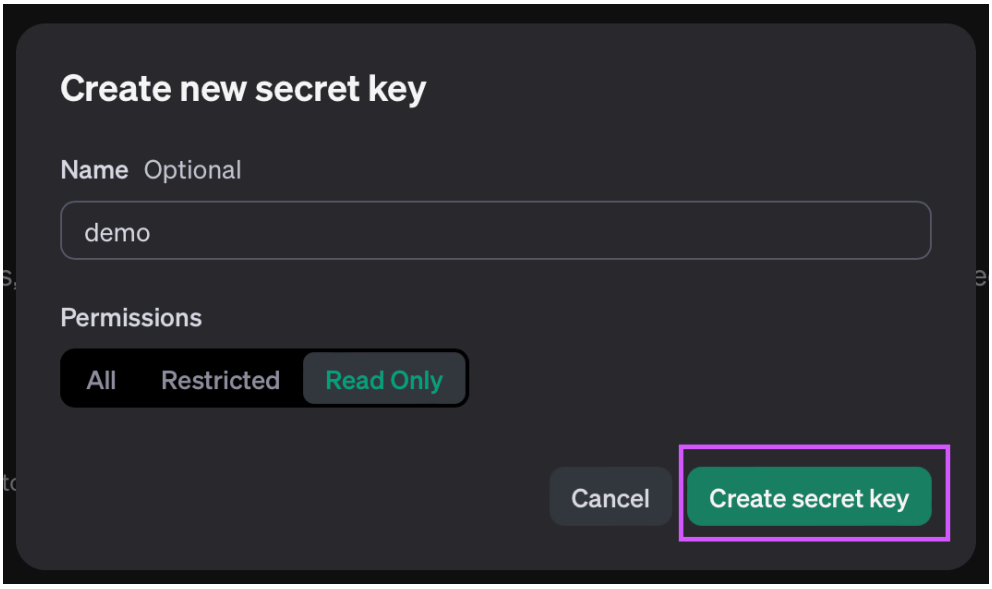

Figure 45. Name the key, ensure you set the permissions scope to “Read Only” then click “Create secret key”.

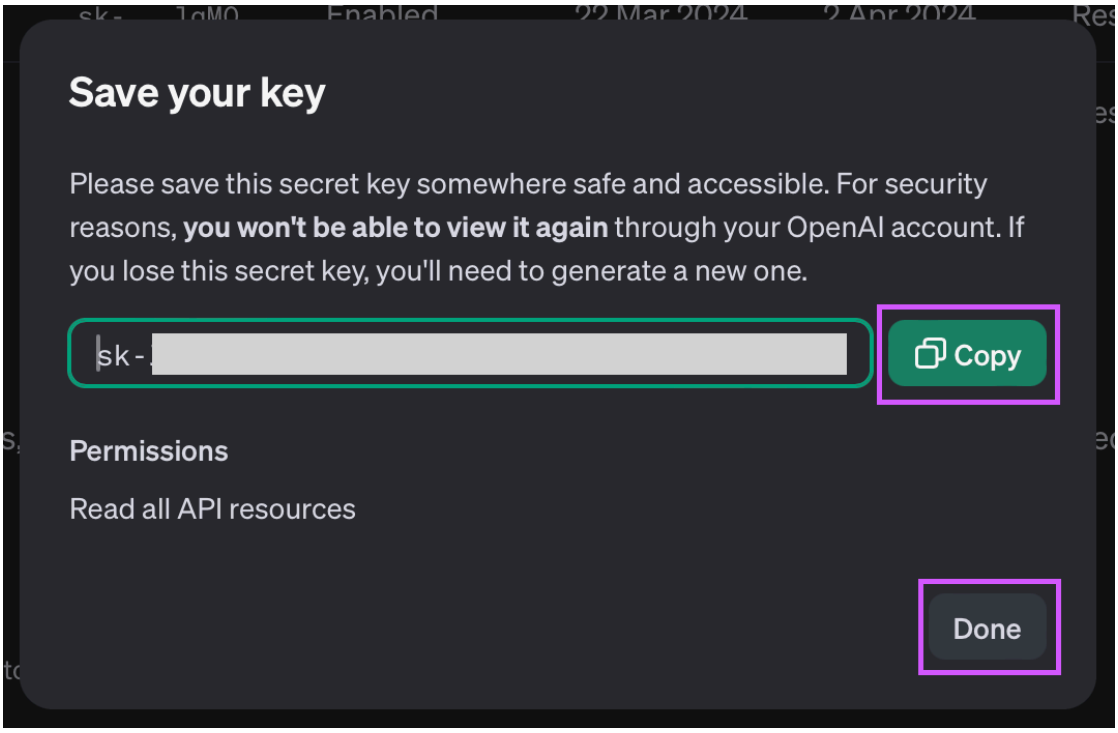

Figure 46. After creating the API key, copy the secret value then click “Done”

Appendix E - Creating Docker Repository

- All the container images that comprise of this project's system are stored in a private Docker repository in my personal account.
- Read-only credentials are provided to access this repository and download the container images within the codebase. Therefore, this step isn't required.
- However, if you wish to create your own repository and host the container images yourself, you can follow this guide.
- The container images' code is located within the "3-containers" directory of the provided codebase.
- Access / create your Docker Hub account here: https://hub.docker.com/signup/
- Download Docker Desktop here: https://www.docker.com/products/docker-desktop/

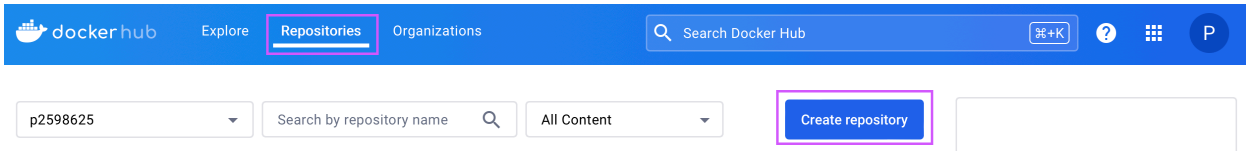

Figure 47. Open the Repositories tab within your Docker Hub account, then click "Create repository".

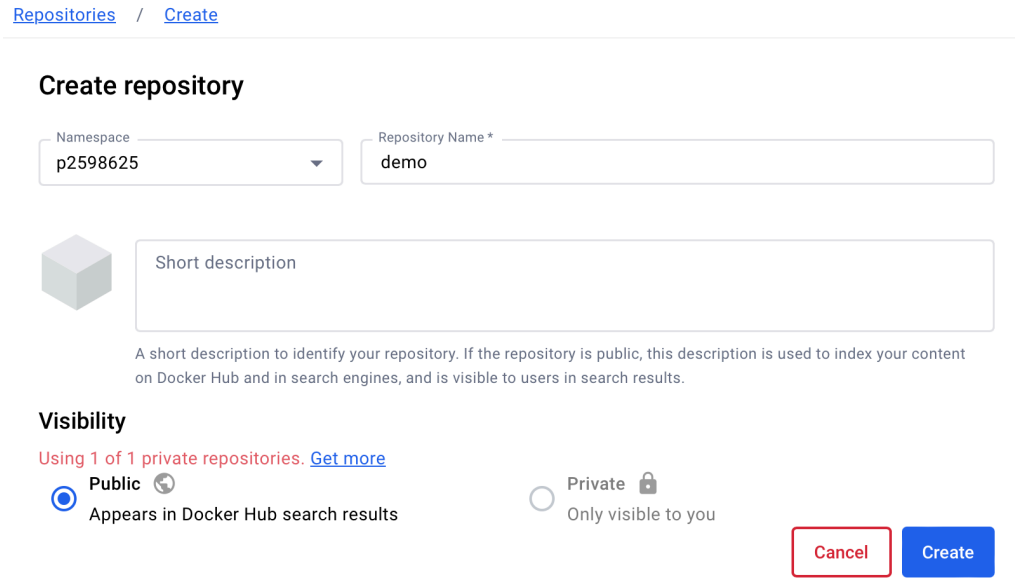

Figure 48. Give the repository a unique name and set the Visibility to Private (this is limited to one per free Docker Hub account hence the error in the screenshot).

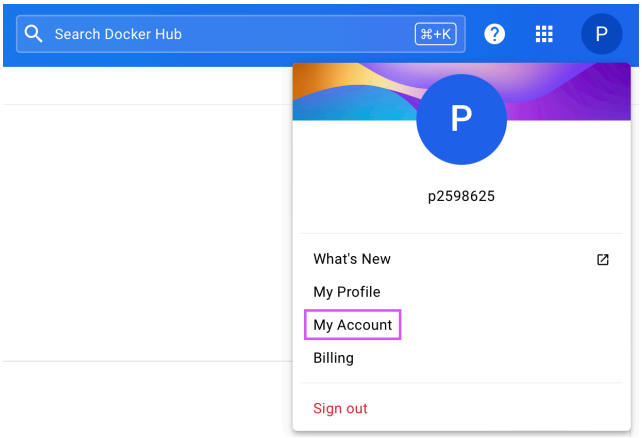

Figure 49. Open your Docker Hub account.

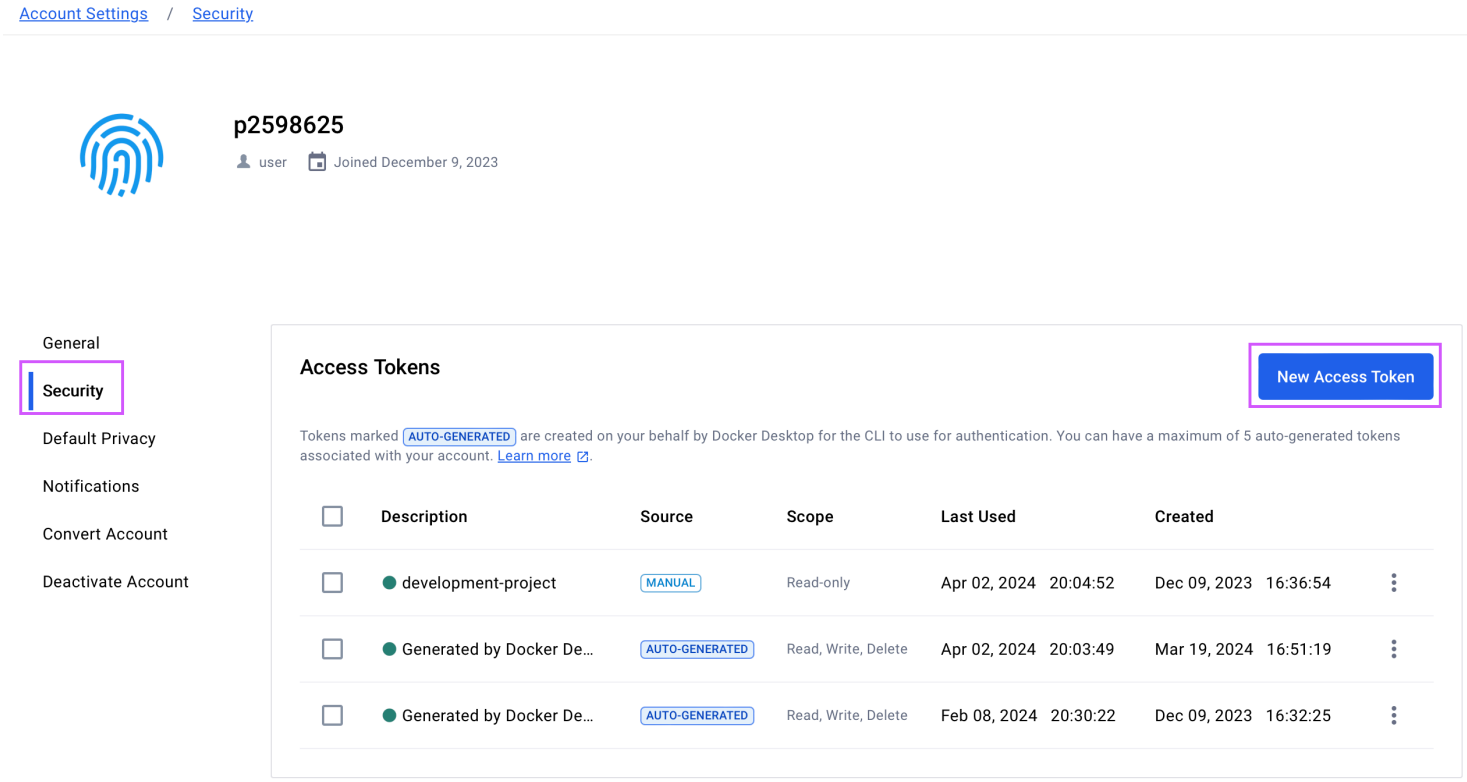

Figure 50. Within the “Security” tab, click “New Access Token”

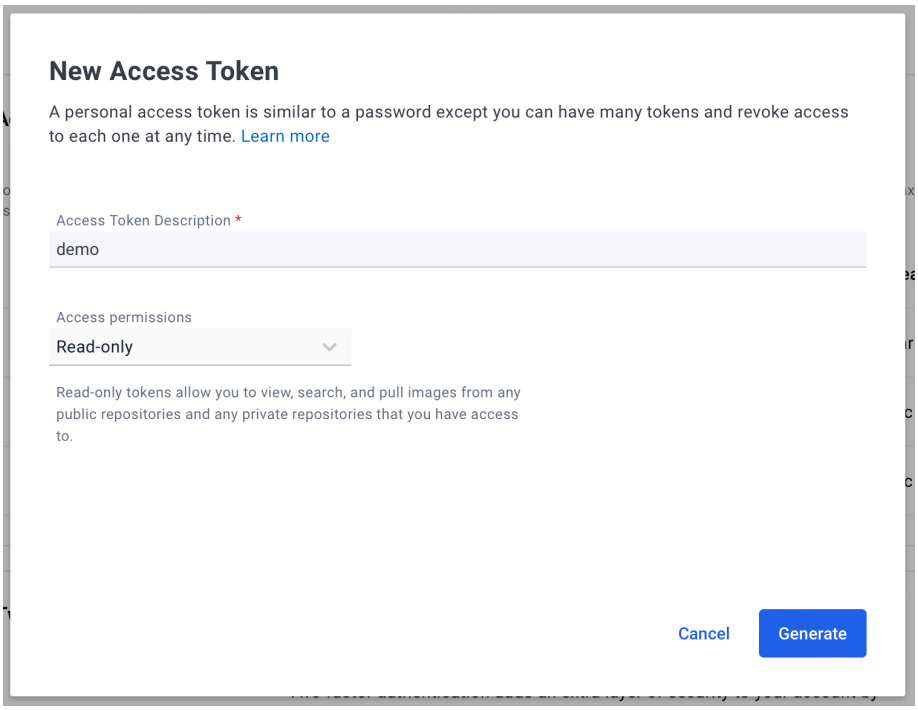

Figure 51. Name the token and set its Access Permissions scope to read-only.

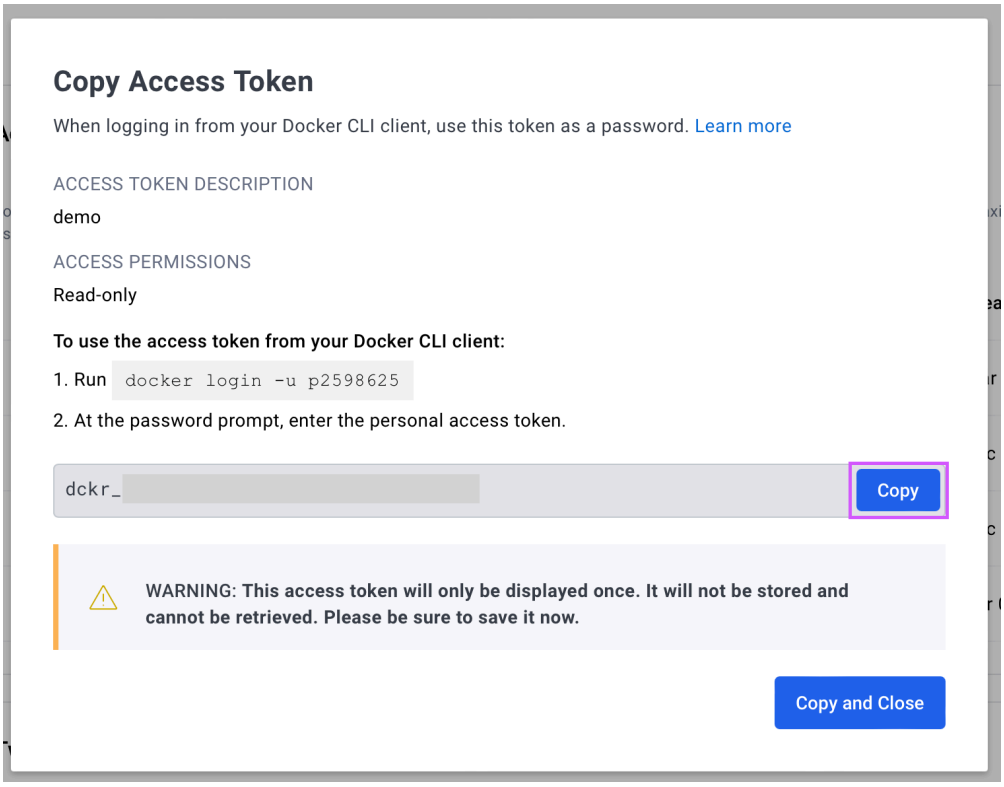

Figure 52. Copy the access token.

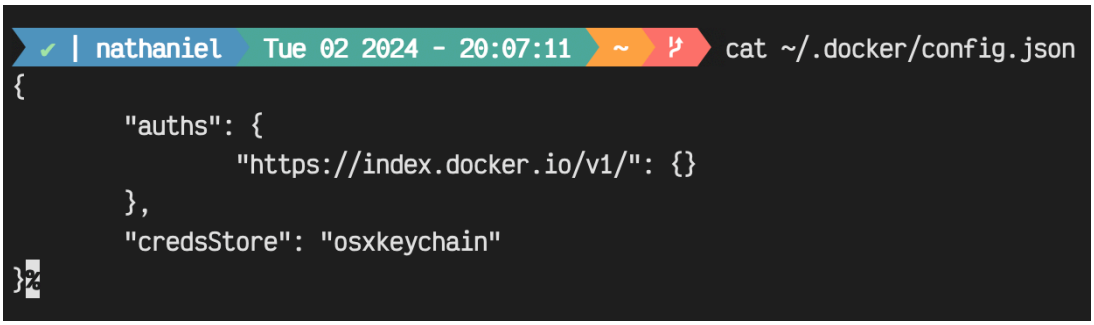

Figure 53. Check the configuration for your Docker Desktop authentication. If you see a "credStore" value, edit "~/.docker/config.json" file and remove the line.

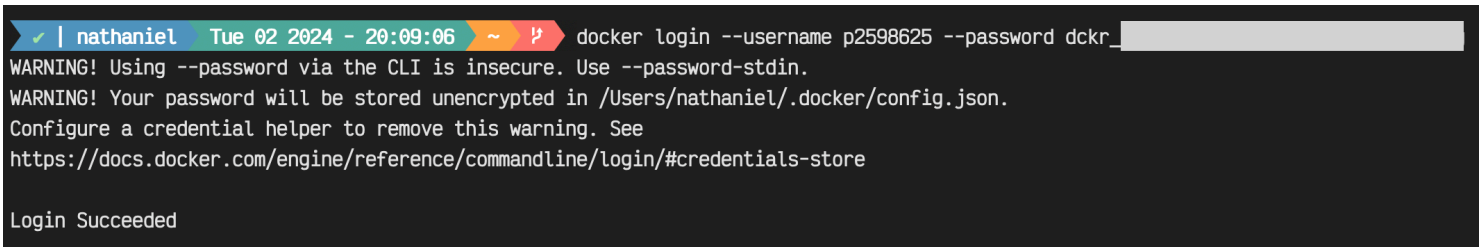

Figure 54. Re-login to Docker Desktop with your username and the access token created in Figure 52 as your password.

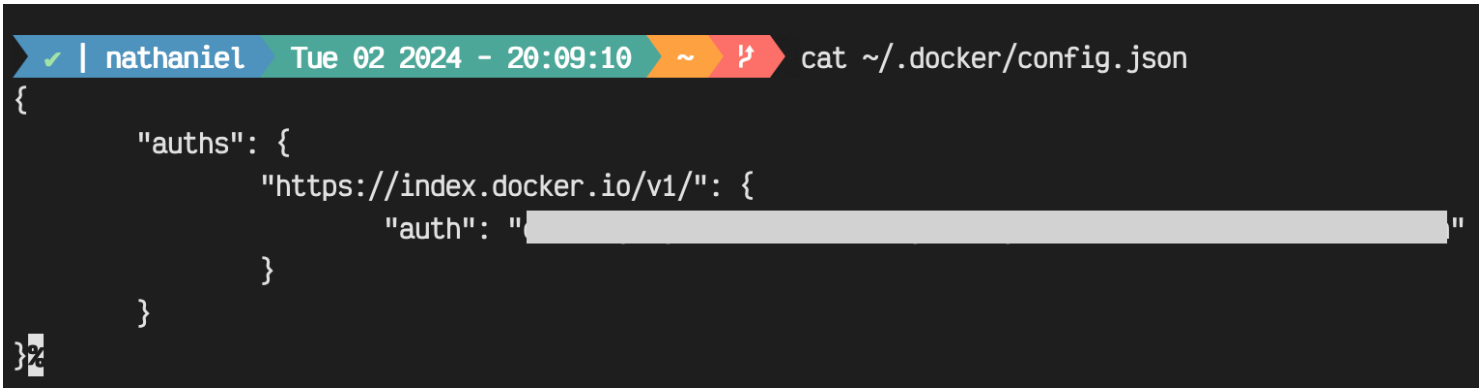

Figure 55. Your Docker Desktop configuration file should now look like this. The "auth" value will be used to authenticate to the Docker repository from within Kubernetes when downloading the container images. See table *User Manual - Secret Values Table* for more details.

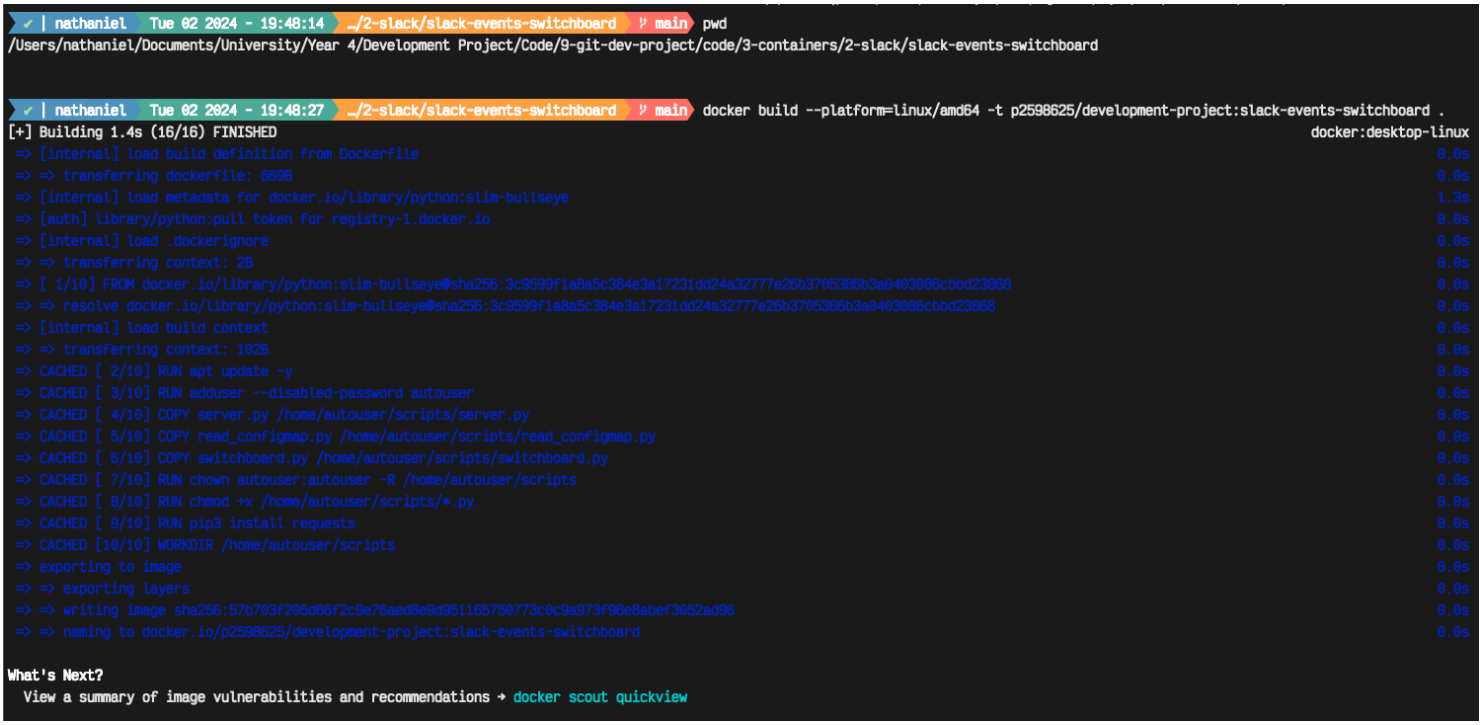

Figure 56. Within a container's code directory (see pwd command for example path), run the docker build command to create a container image.

- *—platform=linux/amd64* This is required when building a container image from a non-amd64 architecture system because the Kubernetes' cluster nodes run amd64 architecture processors. In this case, I'm building the containers from an M1 MacBook which is based on arm64 architecture.

- *-t* This specified the tag you wish to apply to the built container image. The format should follow such: "account name"/"repository name":"container image name".

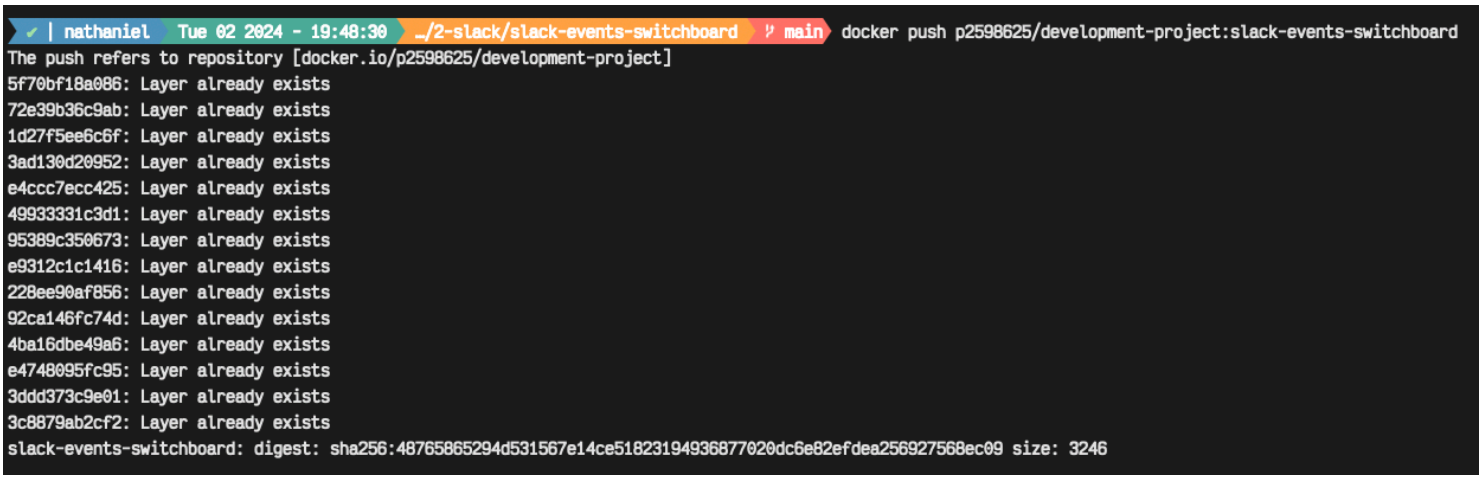

Figure 57. Upload the container image to the destination repository specified within the image's tag.

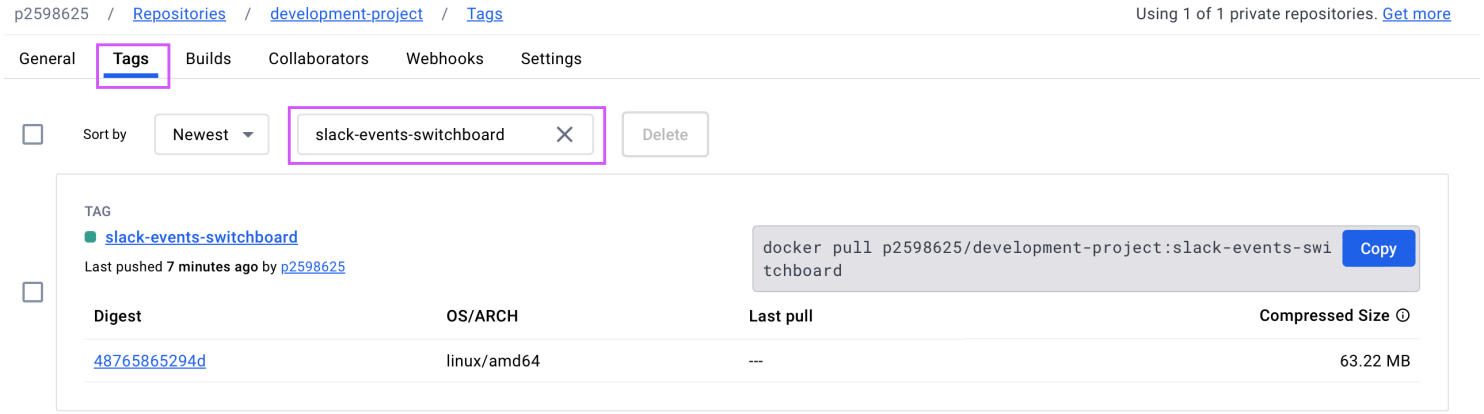

Figure 58. The container image built (see Figure 56) and pushed (see Figure 57) will be visible from your Docker Hub account under the Tags tab. You can search for the image using the "container image name" portion of its tag.

```
9-git-dev-project > code > 2-kubernetes > slack > templates > knative_services.yaml
1  apiVersion: serving.knative.dev/v1
2  kind: Service
3  metadata:
4    name: events-switchboard
5
6  spec:
7    template:
8
9      metadata:
10       annotations:
11         autoscaling.knative.dev/min-scale: "1"
12
13     spec:
14       containers:
15       - name: events-switchboard
16         image: "account name"/"repository name":"container image name"
17         imagePullPolicy: IfNotPresent
18
19         ports:
20         - containerPort: 5555
21           protocol: TCP
22
23         env:
24         - name: RELEASE
25           value: "v0.01"
26
27         - name: SLACK_SIGNING_SECRET
28           valueFrom:
29             secretKeyRef:
30               name: jarvis-slack-signing-secret
31               key: secret
32
33         volumeMounts:
34         - name: "config"
35           mountPath: "/home/autouser/scripts/config"
36           readOnly: true
37
38       volumes:
39       - name: "config"
40         configMap:
41           name: switchboard
42
43       imagePullSecrets:
44       - name: dev-project-registry
45
```

Figure 59. To run the container image within the Kubernetes cluster, you will need to update the relevant Knative Services' *spec.template.spec.containers[o].image* value, as seen in the example screenshot.

Appendix F - Rationale For Using Kubernetes

In recent years, there has been a shift of focus from deploying applications on Virtual Machines (VM) to utilising containers that are orchestrated with Kubernetes (Jaworski, no date). There are many reasons for this, one of which could be the fact that Kubernetes predecessor, Borg, was utilised by Google and then open-sourced as the now refined Kubernetes software (Jaworski, no date). Additionally, containers allow for efficient and dynamic horizontal scaling to meet demand. In the past, virtual machines' resources would be increased inline with their workload and it would take long startup times to provision new VMs. As containers are far smaller in size and share a kernel with their host, they can be deployed far more rapidly with minimal lag-time (Cloud Control, no date).

Due to the new popularity of Kubernetes, I wanted to explore the possibility of hosting security tooling using it. This approach reduces the costs of implementing my project's applications into a new environment as the infrastructure would likely already exist. Furthermore, I was able to exploit native features of Kubernetes such as the automated provisioning and scaling of containers, extended by the functionality of Knative. Likewise, deployment platforms like ArgoCD ease the management of containers via syncing with source control and providing a graphical user interface to visualise the application.

Knative's ability to scale running containers from the amount of HTTP traffic received facilitates the core orchestration of my application. By only running containers when they are needed, the cluster's resources can be efficiently utilised to meet the demand of users. Moreover, the Knative project has adopted an ingress controller, Kourier, to facilitate the control of network traffic between containers and the internet. This made the instalment and configuration much easier as documentation already exists for combining the two as it is by design (Knative, no date - a).

Only two of the Knative services (events-switchboard and interactions-switchboard) are exposed to the internet. The rest are only available from within the cluster as this reduces latency but also improves security by reducing the surface area of possible compromise. To easily encrypt this internal network traffic, I used Linkerd as a service mesh which couples a container alongside my existing applications to proxy traffic. Through routing traffic via the proxies first, encryption can be applied without needing to make any code changes to my applications (Linkerd, no date). The installation of Linkerd was straightforward and easy, with the root TLS certificates created using Terraform.

On the other hand, as Linkerd only encrypts internal network traffic, I used Cert Manager to provision certificates for public-facing Knative services. Paired with lets-encrypt, I was able to create free certificates for my application that were updated automatically when expired (Let's Encrypt, no date). Similarly to Kourier, Knative has existing documentation on how to combine the two (Knative, no date - b). Without encrypting this network traffic, all messages sent to the automation in Slack would be readable from the internet, possibly exposing sensitive information about AWS cloud accounts.

To interact with the different 3rd party services, my application required API tokens to authenticate. These values are intended to be secret and not shared via source code. To this end, I utilised AWS' Paramater Store as a central storage location and then External Secrets to sync the parameter resources into the Kubernetes cluster. This is achieved first by configuring a Secret Store resource which describes how authentication to Parameter store is achieved, in this case an IAM Role for Service Account (AWS, no date - c) is used. Next, External Secrets resources are created to define which values to sync from Parameter Store. These include configuring the name, destination, format and sync schedule of the secret Kubernetes resources. Once the resources are created in Kubernetes, they can be mounted onto containers either as environment variables or as files. The ease of deployment and native support of AWS' parameter store were the main contributing factors to the decision of using External Secrets. Other options such as Bitnami's Sealed Secrets seemed more difficult in installation and configuration and added the additional complexity of encrypting secrets which, in my view, is not an issue with the existing Kubernetes model.

Finally, to host the Kubernetes cluster I chose to utilise AWS' Elastic Kubernetes Service (EKS) with node groups as worker nodes. EKS offers easy provisioning of a Kubernetes cluster with only worker nodes needing to be managed by the customer. AWS will install and configure the control plane of Kubernetes which includes API Server, etcd and scheduler. Three EC2 instances were used as worker nodes which provide the computing resources to run the containers.

Appendix G - Rationale For Using DynamoDB

The majority of my project's application is stateless, no data is persisted within the Kubernetes cluster to ensure any errors do not cascade and lose important information. However, it was required to maintain a record when approval is required for an action. This is because the sending of the approval request and receival of the answer is asynchronous - they occur at different, unknown times. Therefore, I needed to store the approval request so when a user provides their response, the automation can complete the action.

The data needing to be stored is not complex in nature, nor is there a requirement for multiple tables forming relationships. To fulfil my requirements, I chose to utilise AWS' DynamoDB key-value database store. Not only is this option cheap ($1.25 per million writes and $0.25 per million reads along with free 25GB storage allowance (AWS, no date - d)), it does not require SQL to be written as I can retrieve the data using the primary key which is included in the user's response. This simple usage coupled with the very low configuration requirements meant I could very quickly create the datastore and then write Python code to get or put data.

Appendix H - Rationale For Using Systems Manager Parameter Store

There are multiple technologies available to store secrets in a central location. AWS offer both Secrets Manager and Parameter Store whereas Hashicorp have their Vault application. I chose Parameter Store from these options mainly due to it being a far cheaper alternative to Secrets Manager and not requiring the configuration of a virtual machine to run the Vault application. AWS' Secrets Manager charge $0.40 for each secret per month whereas Parameter Store do not charge for their standard parameter types.

Appendix I - Rationale For Using Python

Python is a highly popular programming language with extensive libraries to support 3rd party services and provide additional features (Scarlett, 2023). I have used it substantially both during my time in academia along with my time on placement for creating automation. Therefore, I have the most experience using this language with which I can write code to create a proficient application.

This project's application processes data received via network traffic either from Slack, AWS or the app's other services. To facilitate this I am using the "requests" library to send and receive HTTP traffic. This is an easy to use module which also supports defining headers for setting authentication values and specific HTTP status codes for error handling.

Furthermore, each service has a HTTP server running so it can receive the network traffic. Using the "http.server" library, I was able to run a server listening on the local host interface and handle both GET and POST HTTP requests.

The "boto3" library is incredibly useful for communicating with AWS' API. It incorporates the functionality present in the command line application into a Python library. Through this, I can securely authenticate to AWS and interact with the different services which is a core concept of my project's application.

Conversely, to meet the outlined objectives for this system, I had to write extensive custom code which I tried to keep as modular as possible. For example, any tasks that would be common between services were integrated into their own specific helper service which include ChatGPT, Slack, DynamoDB and S3. Each helper service accepted specific requests to interact with their associated 3rd party service. This reduced the duplication of code and eased maintenance and troubleshooting of errors.

Appendix J - User Manual

This guide will outline the steps to deploy the application to an AWS account and then showcase the different automation actions available.

Prerequisite Requirements

1. AWS Account (costed) — You will need an AWS account to run this application as the infrastructure has been designed to utilise their services. It is recommended to setup either an IAM user or role that has sufficient permissions to create and manage resources within the account who's credentials can be used later. It is also worth noting that AWS do offer a free tier however not all of the services used by this app are included, namely EKS. The Estimated Running Costs table can be used to gauge the running costs for this project's system on a hourly basis (where possible).

2. Slack Account (free) — To interact with the application and create a bot, you will need a Slack account which can be created here: https://chromium.slack.com/signup#/domain-signup. Follow the steps in Appendix A to create a bot.

3. GitHub Account (free) — You will need to create a GitHub account, repository and app so ArgoCD can connect and sync the Kubernetes resources into the cluster. An account can be created here: https://github.com/signup. Follow the steps in Appendix C to create a repository and app.

4. Applications Installed Locally (free) — A few utilities are need to be installed on your local machine so you can build and interact with the application. Some are noted optional as these are only if you wish to interrogate the application in a more technical manner.
    - AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
    - Terraform: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
    - KubeCTL: https://kubernetes.io/docs/tasks/tools/
    - Helm (optional for viewing the installed charts): https://helm.sh/docs/intro/install/
    - Docker (optional for building container images): https://docs.docker.com/engine/install/
    - Visual Studio Code (optional for viewing the codebase in a more friendly format. You can use a different text editor if you have a preference): https://code.visualstudio.com/

User Manual - Estimated Running Costs Table

AWS Service	Resources	Unit Cost	Quantity	Total Cost (per month)
IAM	• Roles • Policies • OIDC Identity Providers	• $0 • $0 • $0	• 10 • 7 • 1	$0
VPC	• Subnets • Internet Gateway	• $0 • $0	• 3 (public) • 1	$0
SSM	• Standard Parameter Store entries	• $0	• 4	$0
DynamoDB	• Table	• $1.25 / $0.25 per 1 million reads / writes[1] • First 25GB free	• < 10 000 read / writes	$0
EC2	• Instances • Classic Load Balancer (created by Kourier)	• $0.0104 / $0.0416 t3 micro / t3 medium per hour • $0.025 per hour	• 3 / 2 • 1	$132
EKS	• Cluster	• $0.10 per hour	• 1	$73
ACM	• Certificate	• $0	• 2	$0
Route53	• Domain Name • Hosted Zone • Records	• Unknown[2] • $0.50 per month • $0	• 1 • 1 • 3	~$0.50
S3	• Bucket	• $0.023 per GB	• 1	$0.02
				Total: $206

[1] It is highly unlikely this application will reach 1 million reads / writes per month. Therefore, I have set the estimated cost to $0.

[2] The cost of your domain name will depend on how in-demand it is.

Deploying Infrastructure

The infrastructure resources are defined in code, written in Terraform. The following steps will outline how to create the AWS services and install the base Kubernetes resources to allow ArgoCD to sync the remainder.

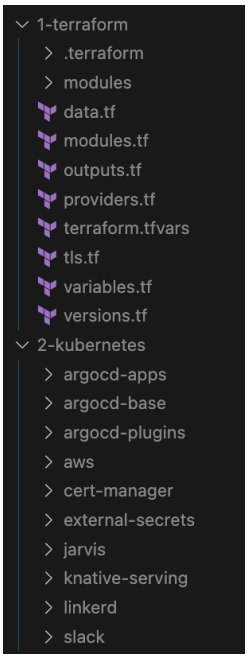

Figure 60. This is the file structure you should have on your local system to install the infrastructure.

Figure 61. Populate secret values in “terraform.tfvars” file located within the “1-terraform” directory. See Secret Values Table for more information.

User Manual - Secret Values Table

Name	Value Source	Reference	Description
argocd_admin_password	This can be set to whatever value you like	N/A	Password of admin ArgoCD user. Used to login to the portal
github_app_private_key	GitHub application's private key	Figure 40	Private key of GitHub app. Used by ArgoCD to authenticate to GitHub and read the repository's files
slack_signing_secret	Slack application's signing secret	Figure 23	Used to authenticate received data from Slack
slack_api_token	Slack application's OAuth token	Figure 24	Used to authenticate to Slack's API endpoint
chatgpt_api_token	ChatGPT API Key	Figure 46	Used to authenticate to ChatGPT's API endpoint
dev_project_registry	My docker hub account.	Appendix E for creating your own registry & credentials	These credentials provide read access to the "p2598625/development-project" repository, allowing container images to be downloaded and ran in the Kubernetes cluster

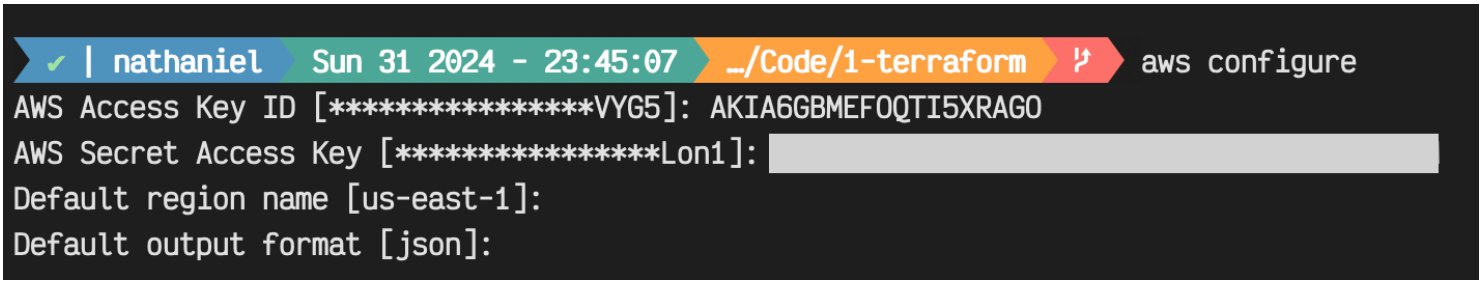

Figure 62. You will need to setup authentication to the target AWS account first. This can be done by using the AWS CLI utility and running "aws configure". You will be prompted to enter your credentials for either an IAM user or role. Terraform will automatically identify these credentials for use when creating / managing resources.

```
✔ | nathaniel  Tue 02 2024 - 10:09:20  …/Code/1-terraform  terraform init


Initializing the backend...
Initializing modules...
- argocd_infra in modules/argocd_infra
- argocd_kubernetes in modules/argocd_kubernetes
- cluster in modules/eks
- demo_resources in modules/demo_resources

Initializing provider plugins...
- Finding hashicorp/tls versions matching "4.0.5"...
- Finding hashicorp/aws versions matching "5.34.0"...
- Finding hashicorp/helm versions matching "2.12.1"...
- Finding hashicorp/kubernetes versions matching "2.25.2"...
- Installing hashicorp/tls v4.0.5...
- Installed hashicorp/tls v4.0.5 (signed by HashiCorp)
- Installing hashicorp/aws v5.34.0...
- Installed hashicorp/aws v5.34.0 (signed by HashiCorp)
- Installing hashicorp/helm v2.12.1...
- Installed hashicorp/helm v2.12.1 (signed by HashiCorp)
- Installing hashicorp/kubernetes v2.25.2...
- Installed hashicorp/kubernetes v2.25.2 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Figure 63. Initialise the Terraform directory by running “terraform init”. This will install the referenced modules and providers along with setting up the state files. The providers have been pinned to specific versions within the “versions.tf” file to reduce the likelihood of newer releases causing breaking changes.

```
✔ | nathaniel  Sun 31 2024 - 23:57:02  …/Code/1-terraform  terraform apply
```

Figure 64. Begin building the resources using Terraform by running “terraform apply”.

```
Plan: 62 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + argocd_url  = "argocd.975050124193.realhandsonlabs.net"
  + knative_url = "*.knative.975050124193.realhandsonlabs.net"

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Figure 65. The command in Figure 64 will run a plan, this should be the output. It will detail the intended number of resources to build, change and destroy along with output values for the ArgoCD portal. Type “yes” to continue, it will take an estimated 20 minutes to build all the infrastructure.

```
Error: services "kourier" not found

  with module.argocd_kubernetes.data.kubernetes_service.kourier,
  on modules/argocd_kubernetes/data.tf line 13, in data "kubernetes_service" "kourier":
  13: data "kubernetes_service" "kourier" {
```

Figure 66. This is a known error caused by a race condition between ArgoCD being installed and running and Terraform trying to find a Service resource.

```
✔ | nathaniel  Tue 02 2024 - 12:26:54  ~  aws eks update-kubeconfig --name jarvis --region us-east-1
Updated context arn:aws:eks:us-east-1:381492166919:cluster/jarvis in /Users/nathaniel/.kube/config

✔ | nathaniel  Tue 02 2024 - 12:27:01  ~  kubectl -n knative-serving get pods
NAME                                      READY   STATUS      RESTARTS   AGE
3scale-kourier-gateway-9bd7579-vxr6v      1/1     Running     0          34s
activator-58db57894b-qxk7k                1/1     Running     0          38s
autoscaler-76f95fff78-lmkkw               1/1     Running     0          37s
autoscaler-hpa-85696784dd-wfd5r           1/1     Running     0          35s
controller-7dd875844b-xwg49               1/1     Running     0          37s
knative-operator-6d5d9b4fdd-mgwwm         1/1     Running     0          54s
net-certmanager-controller-84f4fc6d5c-49cnd   1/1     Running     0          43s
net-certmanager-webhook-5776b94c64-tdx65  1/1     Running     0          43s
net-kourier-controller-85dfb7ccbc-8w98w   1/1     Running     0          34s
operator-webhook-6b67777656-r9mwg         1/1     Running     0          54s
storage-version-migration-serving-serving-1.13.1-2hk6h   0/1     Completed   0          35s
webhook-d8674645d-tqh4w                   1/1     Running     0          36s
```

Figure 67. To fix the error seen in Figure 66, wait for ArgoCD to sync all Knative resources so the Kourier deployment is running. You can use the aws and kubectl applications to check this, as seen in the screenshot, it will take an estimated 10 minutes for ArgoCD to complete the rollout.

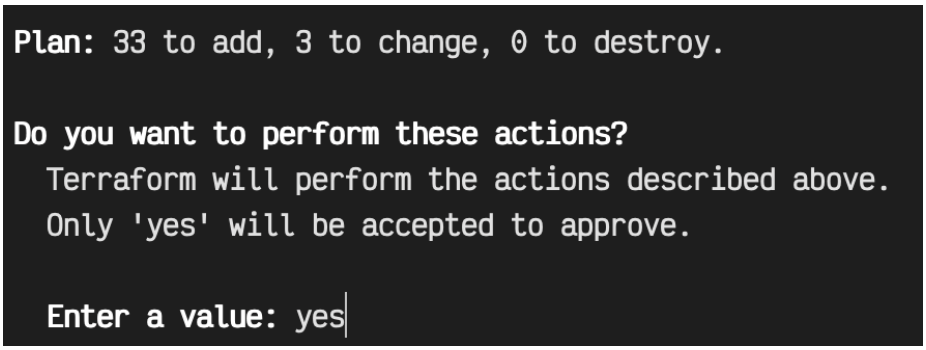

Figure 68. Once the Kourier deployment is running, you can re-run "terraform apply" (see Figure 64) to continue deploying the remaining infrastructure. Enter "yes" to make the planned changes.

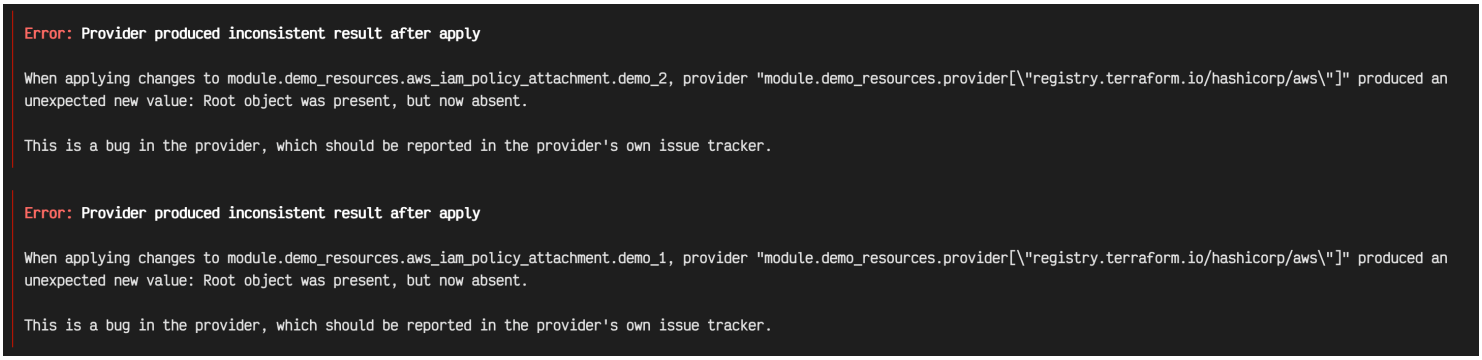

Figure 69. This too is a known error with the AWS Terraform provider. It can be ignore and fixed by just re-running "terraform apply" again.

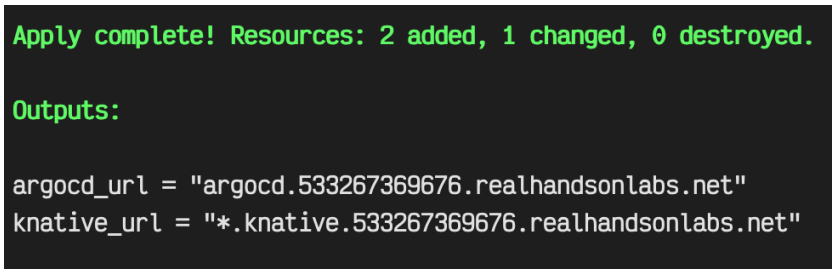

Figure 70. Once all the infrastructure has been deployed successfully, you will receive the following output from Terraform.

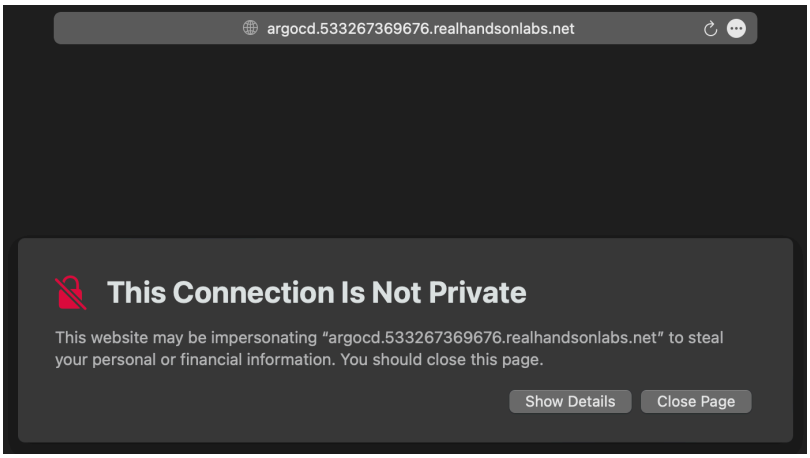

Figure 71. Within your browser, go to the "argocd_url" value seen in Figure 70. The warning seen in the screenshot can be ignored as it is caused by the default TLS certificate for the URL being self-signed. It will take some time for the Certificate created in AWS to propagate.

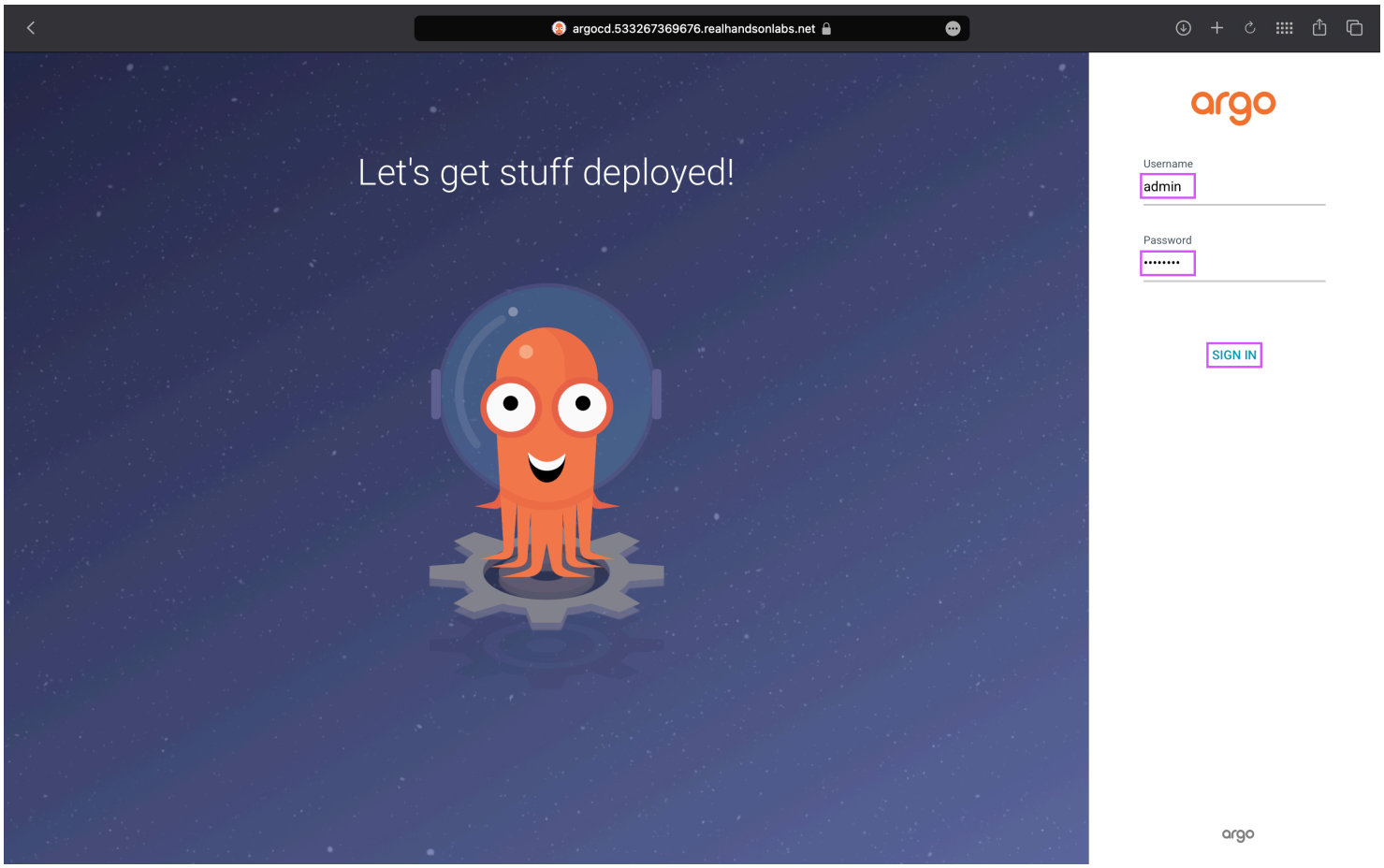

Figure 72. Enter "admin" as the username and the value you set for "argocd_admin_password" in terraform.tfvars file (see Figure 61) for the password. The click Sign In.

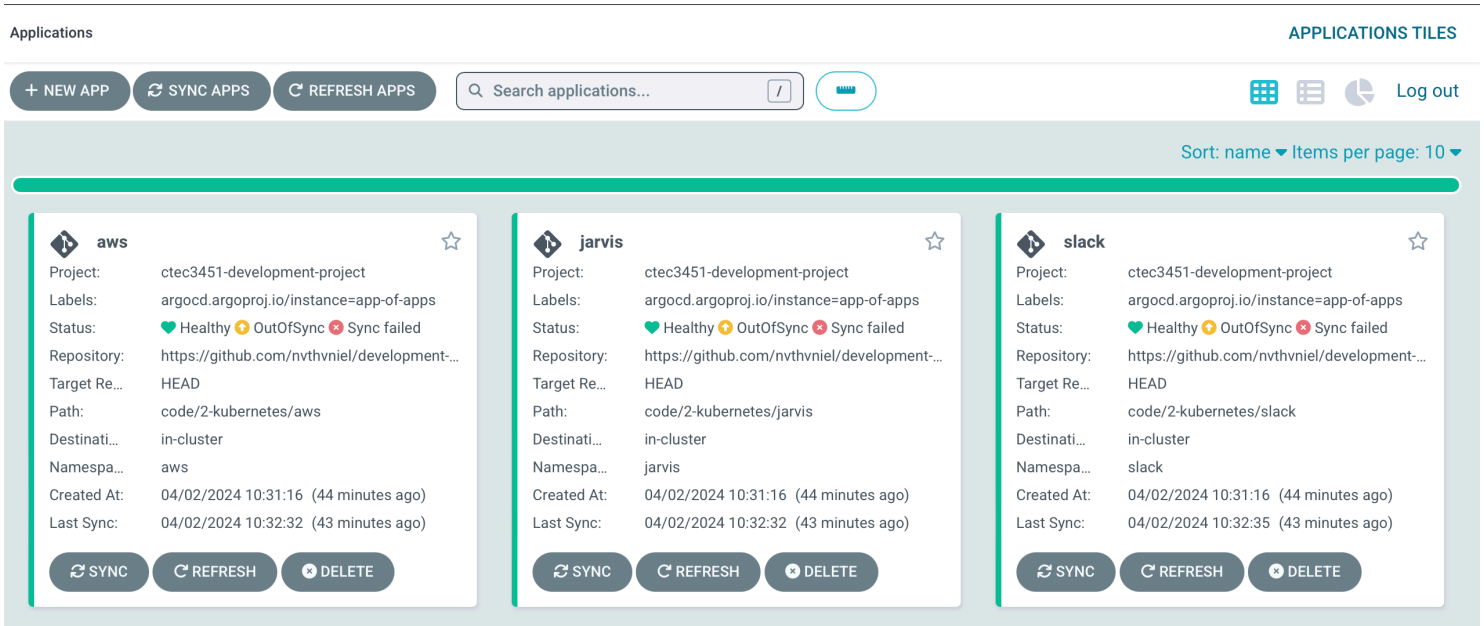

Figure 73. If any applications' status show as "OutOfSync" or "Sync failed", click the "Sync" button then "Synchronize". These are usually caused by a race condition between Knative being fully installed and the above applications trying to use Knative resources.

```
✔ | nathaniel  Tue 02 2024 - 12:37:32  ~  kubectl -n slack get routes
NAME                    URL                                                                  READY   REASON
events-switchboard      https://events-switchboard.slack.knative.381492166919.realhandsonlabs.net   True
helper                  http://helper.slack.svc.cluster.local                                True
interactions-switchboard https://interactions-switchboard.slack.knative.381492166919.realhandsonlabs.net   True
```

Figure 74. Once all applications have been successfully synced in ArgoCD, retrieve the switchboard URLs using the kubectl command shown in the screenshot. Cert-manager will automatically provision TLS certificates to facilitate HTTPS communications.

- Set your slack app's Event Subscription Request URL (see Figure 26) to the URL of "events-switchboard" shown in Figure 74.

- Set your slack app's interactivity Request URL (see Figure 27) to the URL of "interactions-switchboard" shown in Figure 74.

Once the URL endpoints for both switchboards have been set in Slack, the configuration of infrastructure is complete.

Sending Commands to Bot

Within your slack channel with the Slack application installed (see Appendix B), you can send commands to make changes in a target AWS account.

Ensure you login as your "responder" user as they won't be able to approver their own requests.

Help —

- Description: outputs help menu, listing all supported actions with their associated declarative prompt format.
- Example Declarative Prompt: *@jarvis help*
- Example Natural Language Prompt: *Not supported*

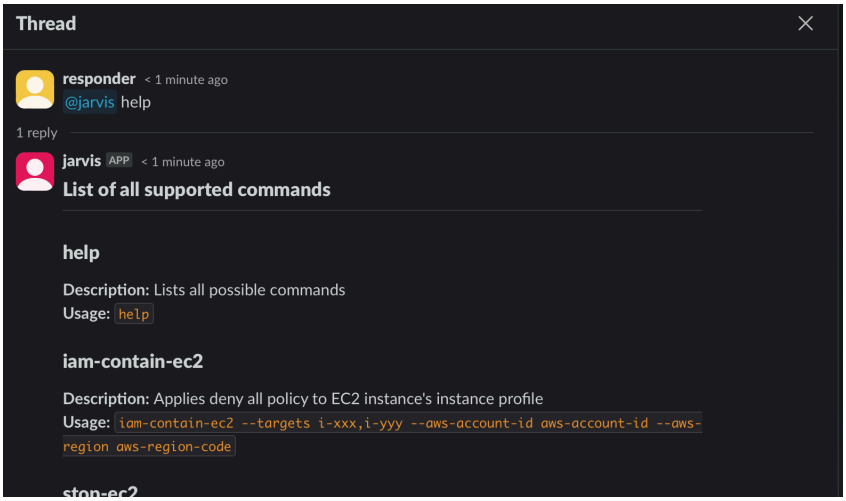

Figure 75. Output from help command.

Refresh User Session —

- Description: Applies policy to IAM user to invalidate associated temporary credentials.
- Example Declarative Prompt: *@jarvis refresh-user-session —targets "user name" —aws-account-id "aws account id"*
- Example Natural Language Prompt: *@jarvis refresh jarvis-demo-no-approval user's temporary credentials in 381492166919*

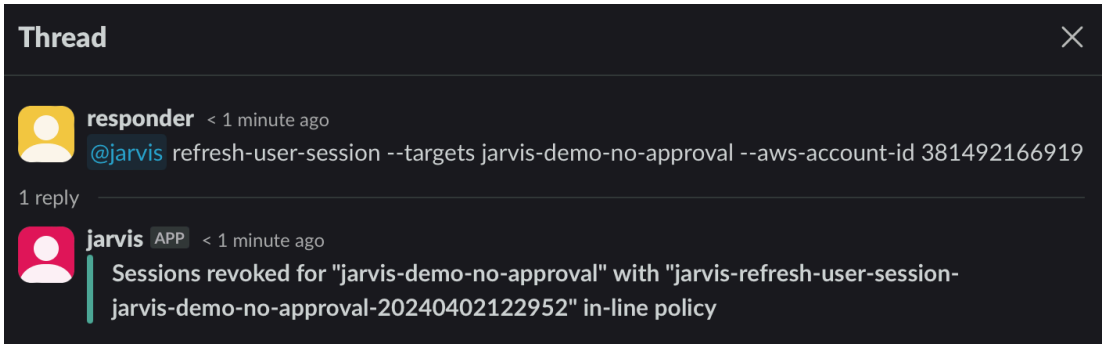

Figure 76. Output from refresh user session command using declarative prompt.

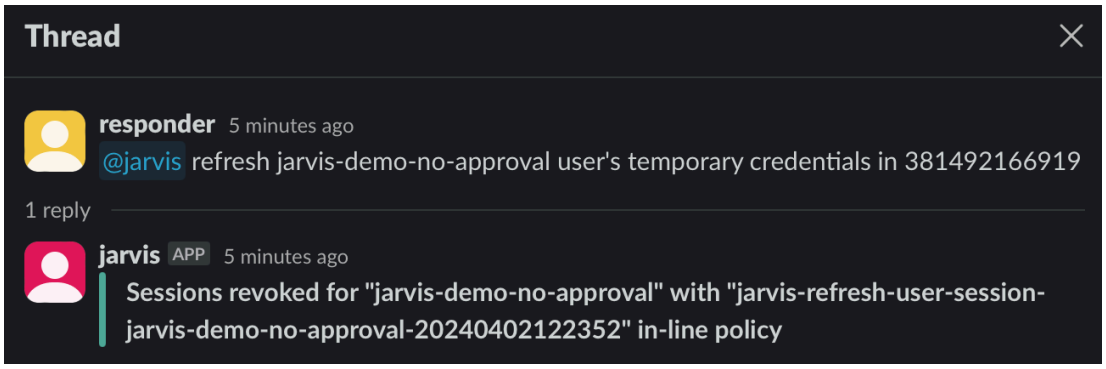

Figure 77. Output from refresh user session command using natural language prompt.

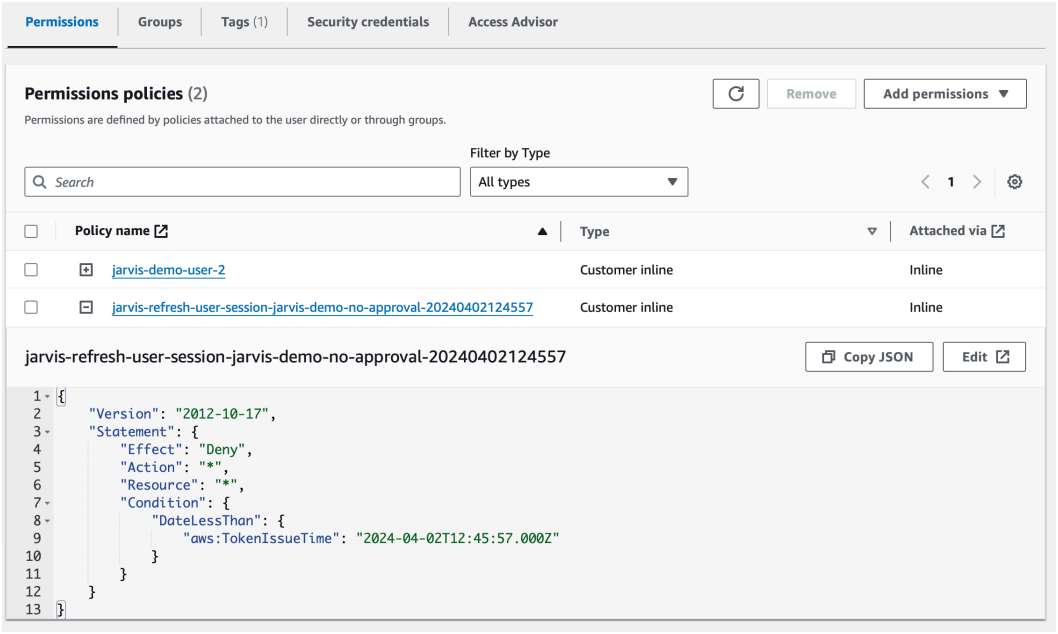

Figure 78. Inline policy added to IAM user in AWS. The policy's condition will deny all actions where the token's issue time is less than the response action's

Refresh Role Session —

- Description: Applies policy to IAM role to invalidate associated temporary credentials.
- Example Declarative Prompt: *@jarvis refresh-role-session —targets "role name" —aws-account-id "aws account id"*
- Example Natural Language Prompt: *@jarvis refresh jarvis-demo-no-approval role's temporary credentials in 381492166919*

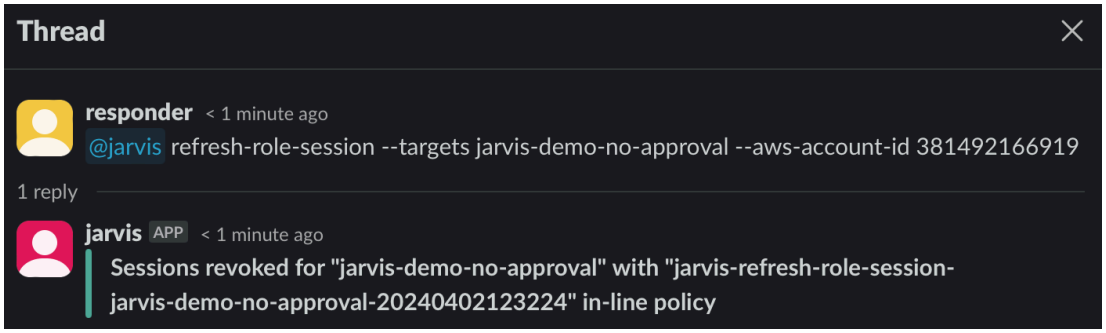

Figure 79. Output from refresh role session command using declarative prompt.

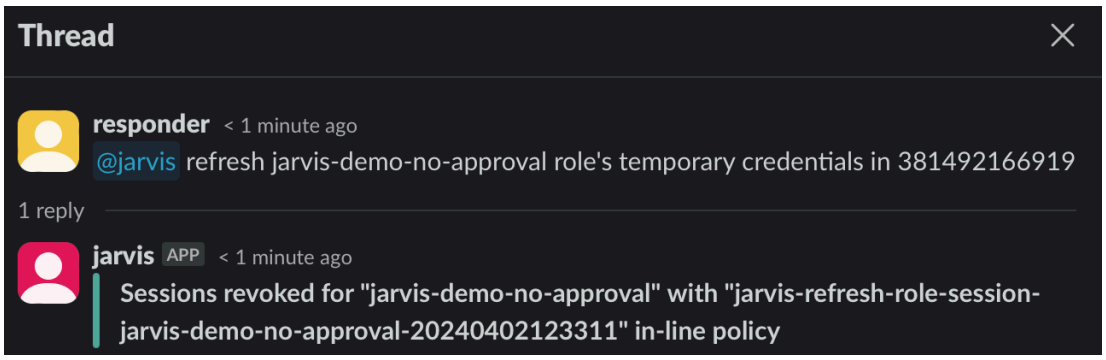

Figure 80. Output from refresh role session command using natural language prompt.

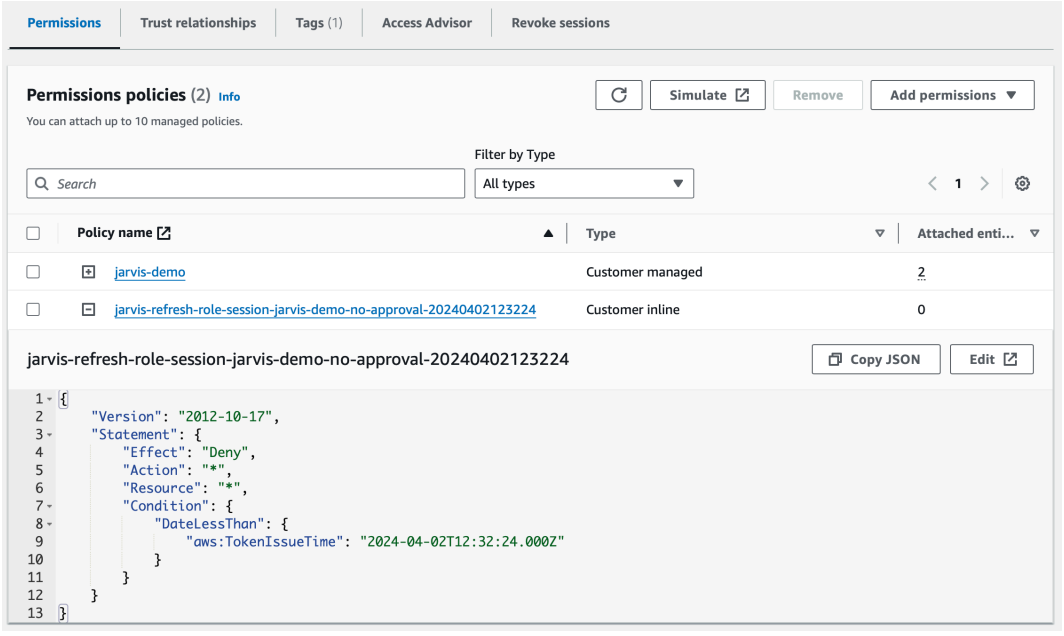

Figure 81. Inline policy added to IAM role in AWS. The policy's condition will deny all actions where the token's issue time is less than the response action's

Disable User —

- Description: Applies deny all policy to IAM user
- Example Declarative Prompt: *@jarvis disable-user —targets "user name" —aws-account-id "aws account id"*
- Example Natural Language Prompt: *@jarvis disable the user, jarvis-demo-no-approval in 381492166919*

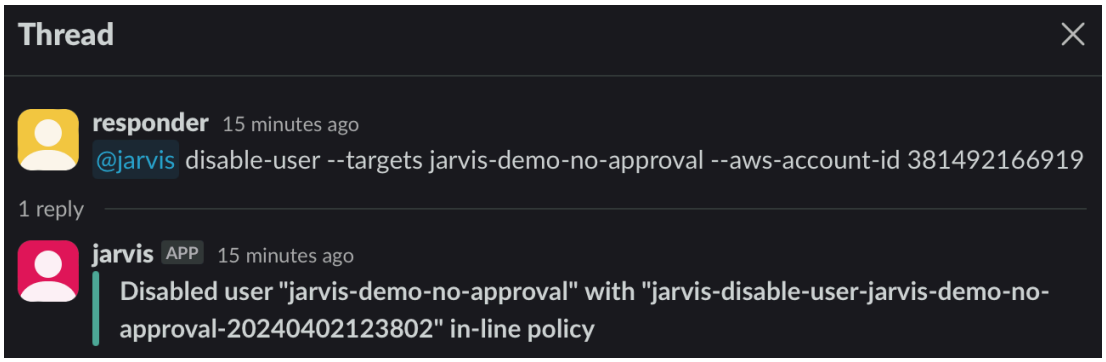

Figure 82. Output from disable user command using declarative prompt.

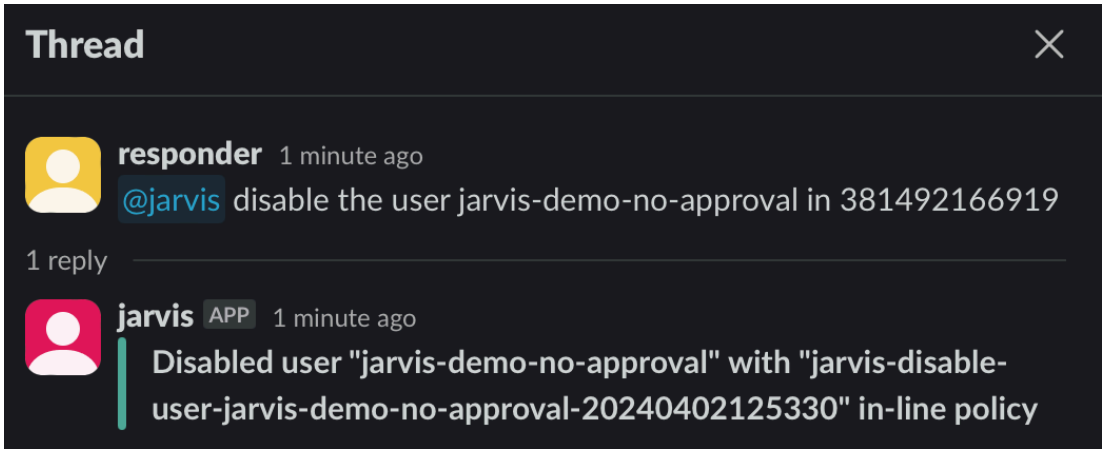

Figure 83. Output from disable user command using natural language prompt.

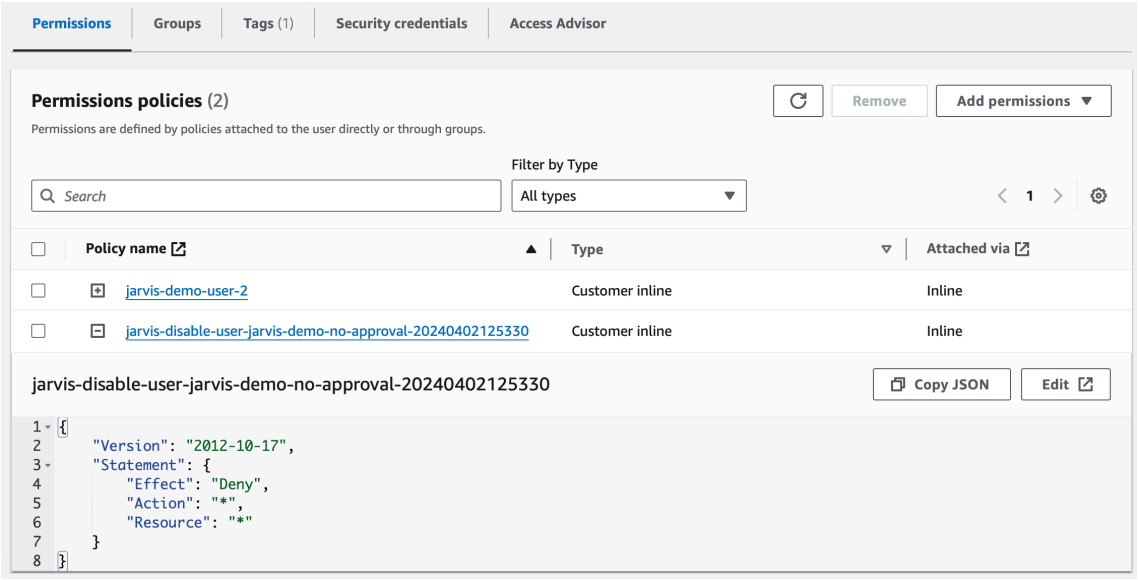

Figure 84. Inline policy added to IAM user in AWS. The policy will deny all actions.

Disable Role —

- Description: Applies deny all policy to IAM role
- Example Declarative Prompt: *@jarvis disable-role —targets "role name" —aws-account-id "aws account id"*
- Example Natural Language Prompt: *@jarvis disable the role, jarvis-demo-no-approval in 381492166919*

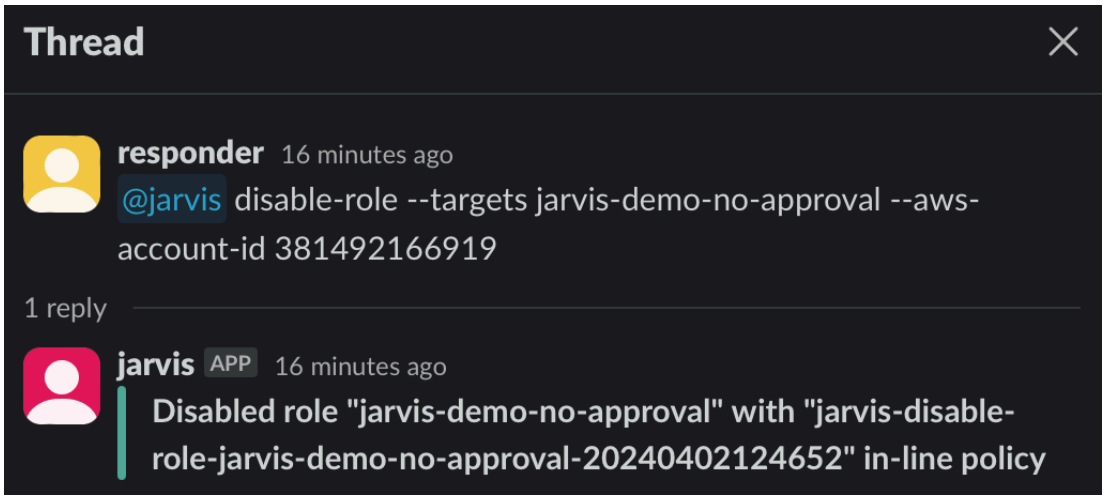

Figure 85. Output from disable role command using declarative prompt.

Figure 86. Output from disable role command using natural language prompt.

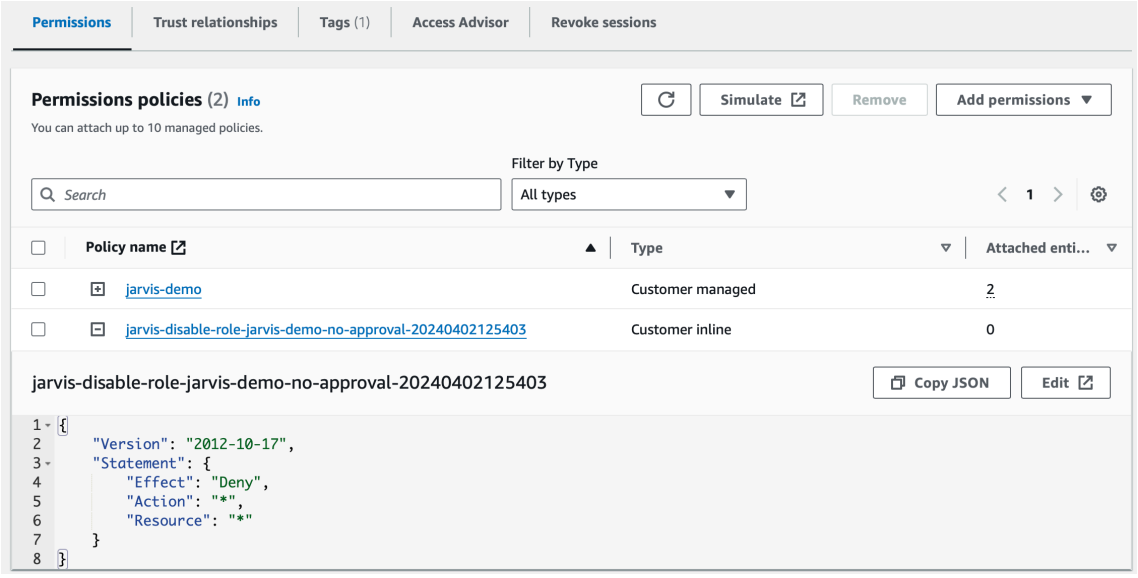

Figure 87. Inline policy added to IAM role in AWS. The policy will deny all actions.

Network Contain EC2 —

- Description: Applies deny all security group to EC2 instance
- Example Declarative Prompt: *@jarvis network-contain-ec2 —targets "instance ID" —aws-account-id "aws account id" —aws-region "aws region code"*
- Example Natural Language Prompt: *@jarvis block network access to i-0e86421aa02b2a869 in us-east-1 381492166919*

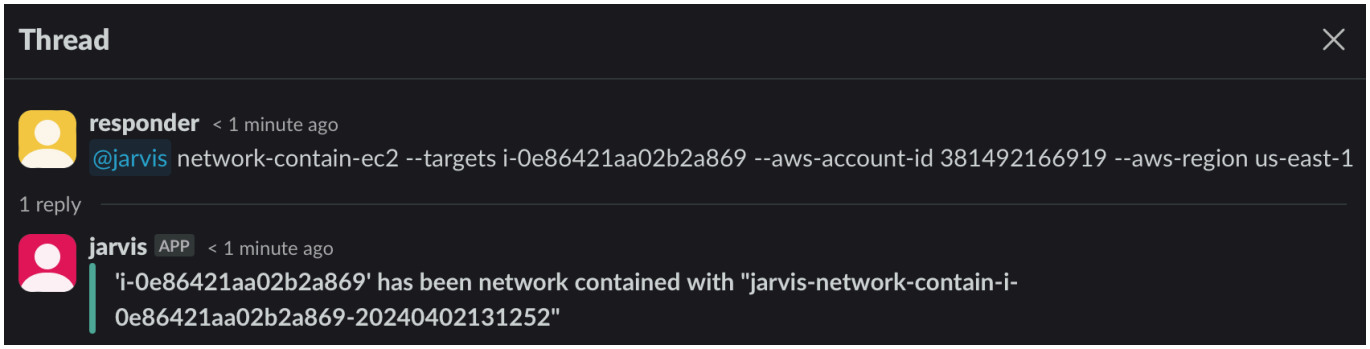

Figure 88. Output from network contain EC2 command using declarative prompt.

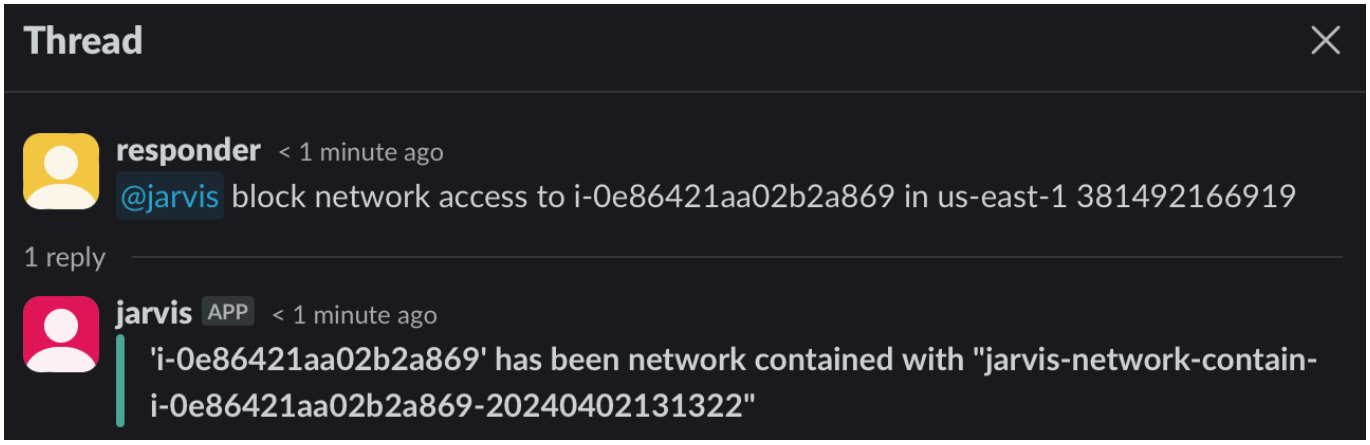

Figure 89. Output from network contain EC2 command using natural language prompt.

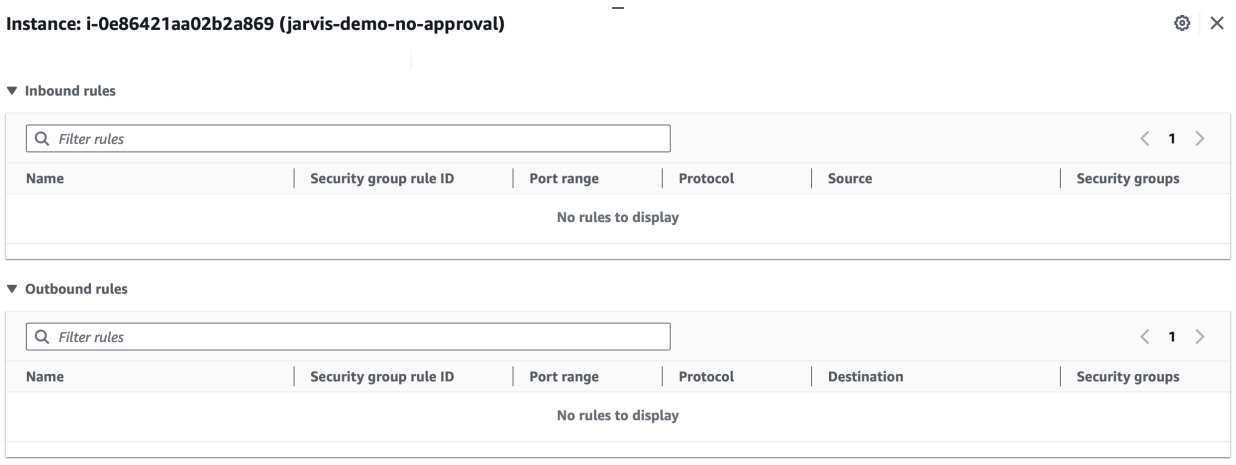

Figure 90. Security group added to EC2 instance. Security groups require explicit definition of what traffic to allow. No rules mean no ingress / egress.

IAM Contain EC2 —

- Description: Applies deny all policy to EC2 instance's instance profile (IAM Role)
- Example Declarative Prompt: *@jarvis iam-contain-ec2 —targets "instance ID" —aws-account-id "aws account id" —aws-region "aws region code"*
- Example Natural Language Prompt: *@jarvis block i-0e86421aa02b2a869 from interacting with the AWS API in us-east-1 381492166919*

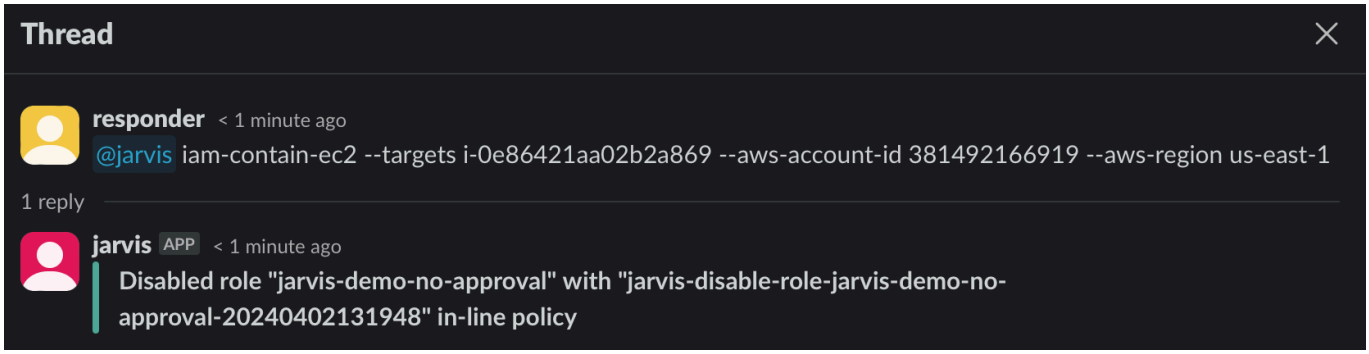

Figure 91. Output from IAM contain EC2 command using declarative prompt.

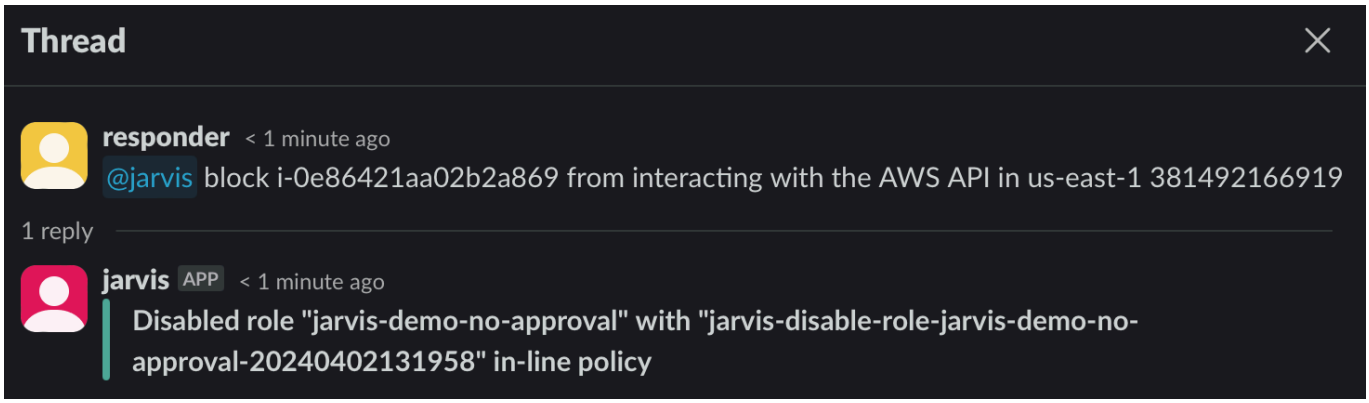

Figure 92. Output from IAM contain EC2 command using natural language prompt.

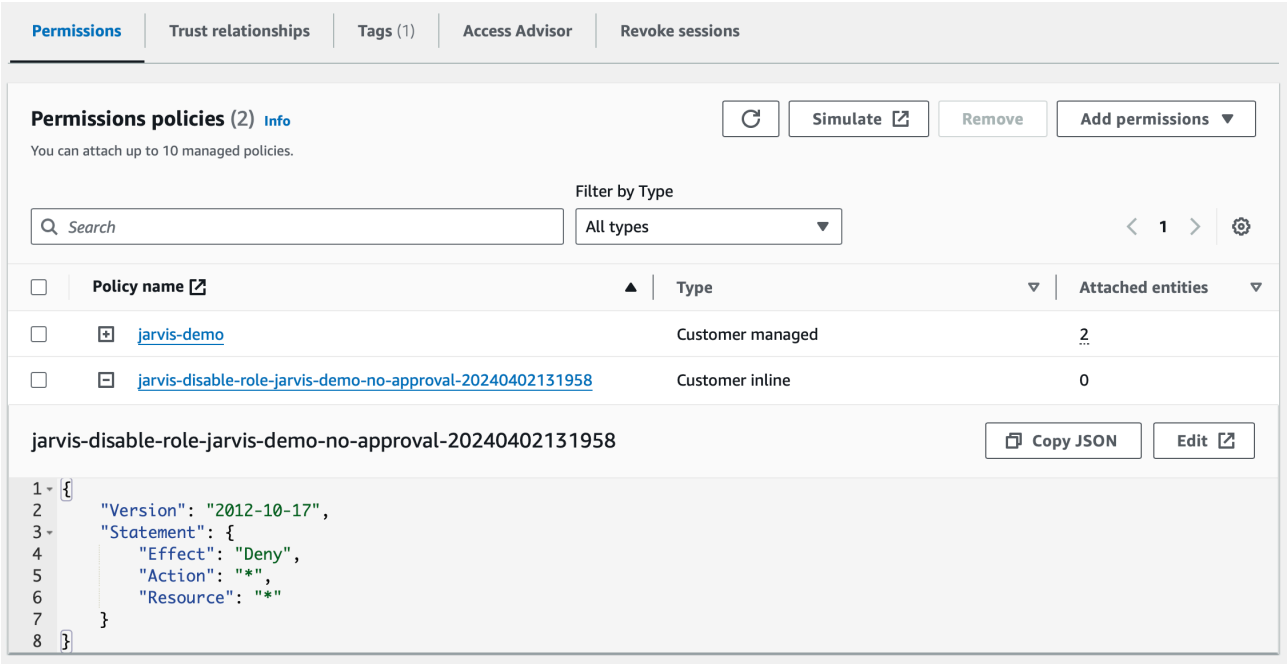

Figure 93. Inline policy added to IAM role in AWS. The policy will deny all actions.

Snapshot EC2 —

- Description: Creates snapshot of all storage volumes attached to EC2 instance
- Example Declarative Prompt: *@jarvis snapshot-ec2 —targets "instance ID" —aws-account-id "aws account id" —aws-region "aws region code"*
- Example Natural Language Prompt: *@jarvis take a snapshot of i-0e86421aa02b2a869 in us-east-1 381492166919*

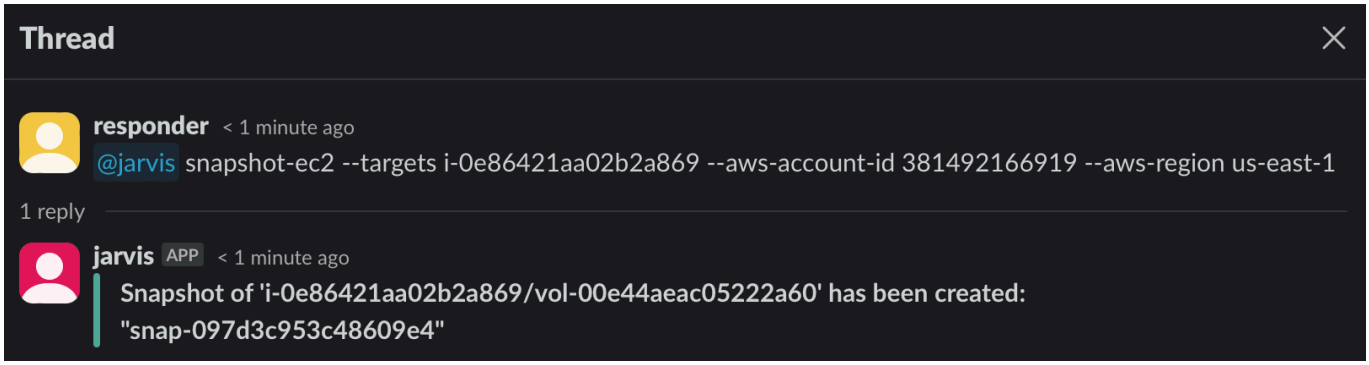

Figure 94. Output from snapshot EC2 command using declarative prompt.

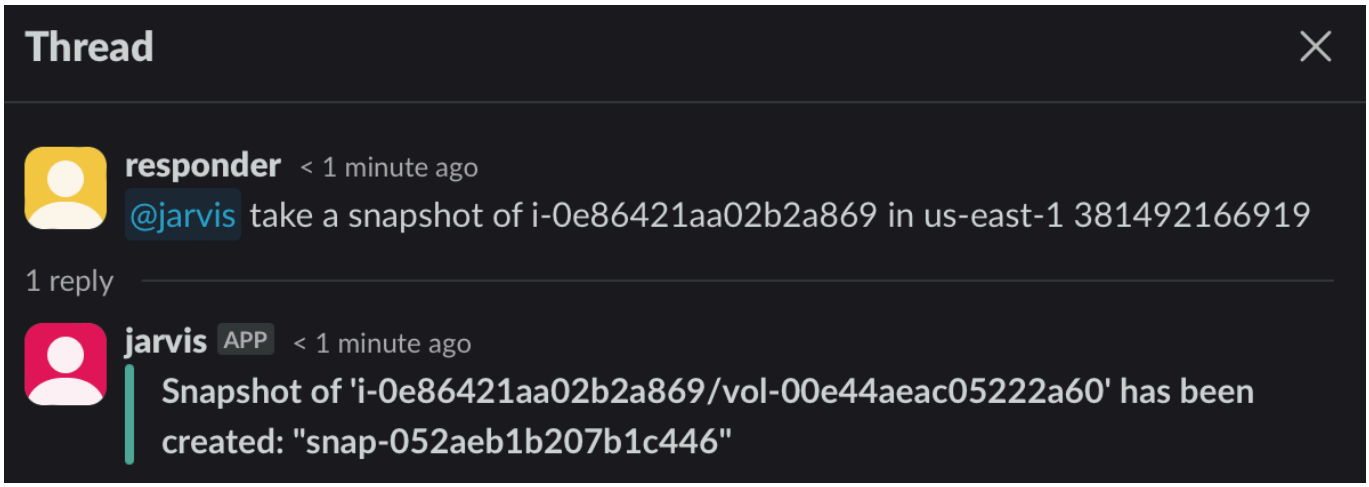

Figure 95. Output from snapshot EC2 command using natural language prompt.

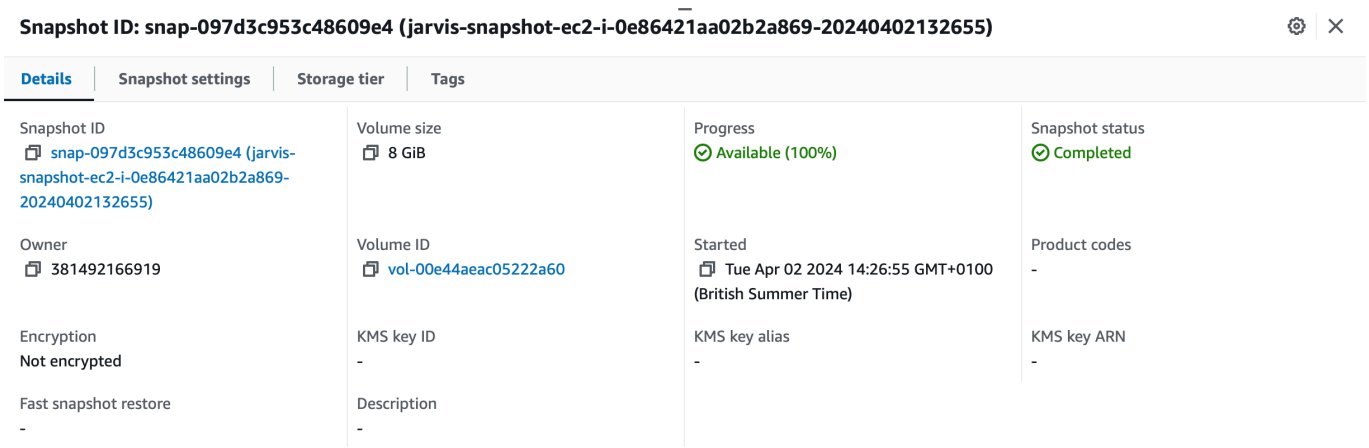

Figure 96. Snapshot of EC2 instance's storage volume created in AWS.

Stop EC2 —

- Description: Stops EC2 instance
- Example Declarative Prompt: *@jarvis stop-ec2 —targets "instance ID" —aws-account-id "aws account id" —aws-region "aws region code"*
- Example Natural Language Prompt: *@jarvis stop i-0e86421aa02b2a869 in us-east-1 381492166919*

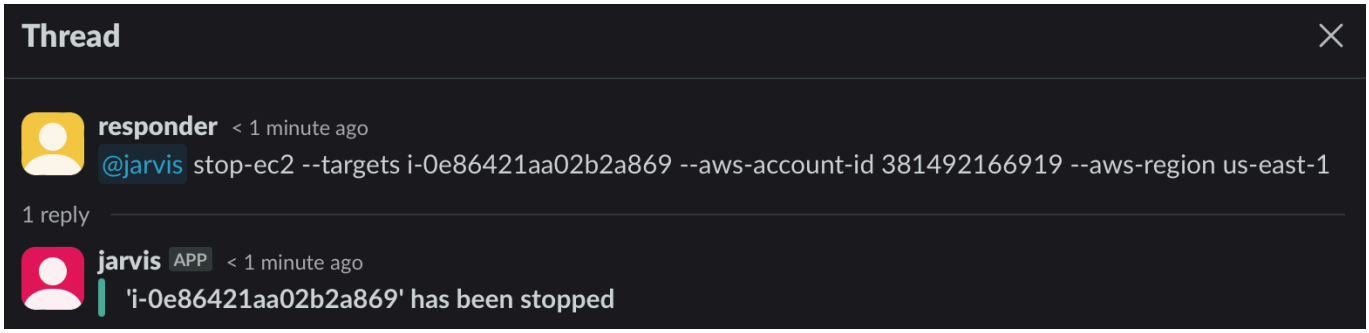

Figure 97. Output from stop EC2 command using declarative prompt.

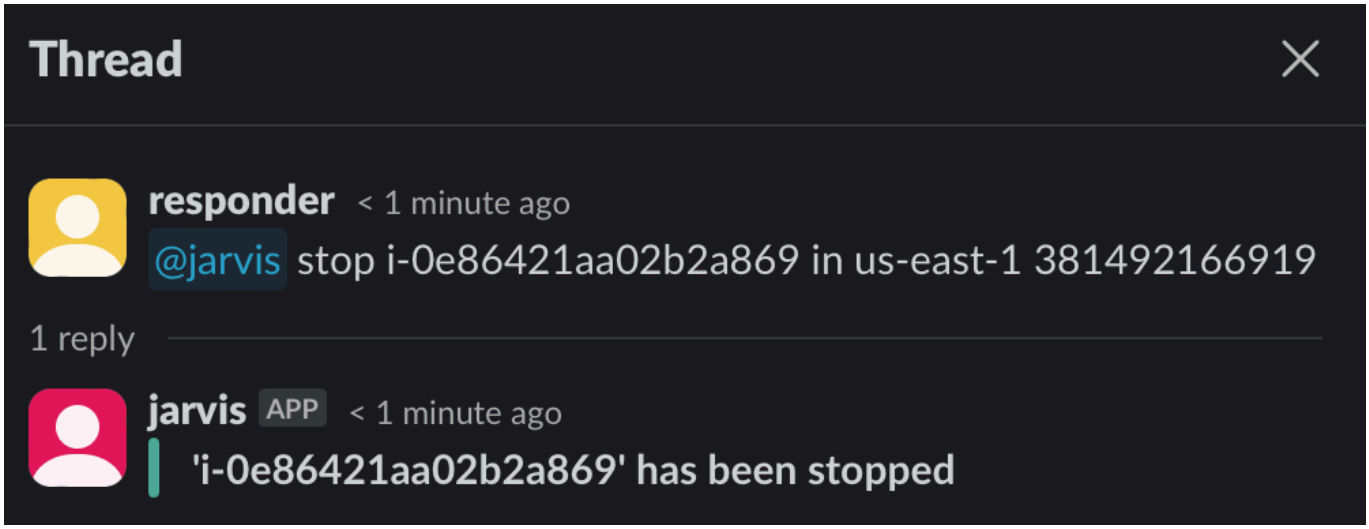

Figure 98. Output from stop EC2 command using natural language prompt.

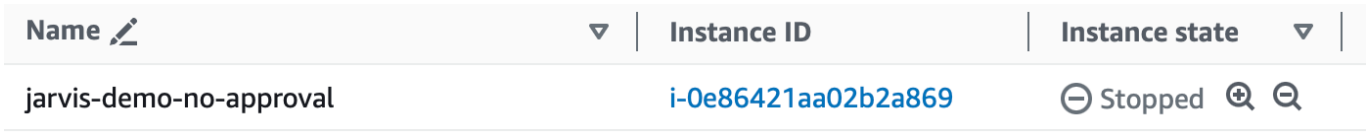

Figure 99. EC2 instance stopped in AWS.

Approval Process —

- **Description:** actions can require approval before changes are made within an AWS account. Either the resource can be tagged with "jarvis-approval-required:true" (see Figure 100) or the automation service can have its "require_approval" value set to "true" (see Figure 101).

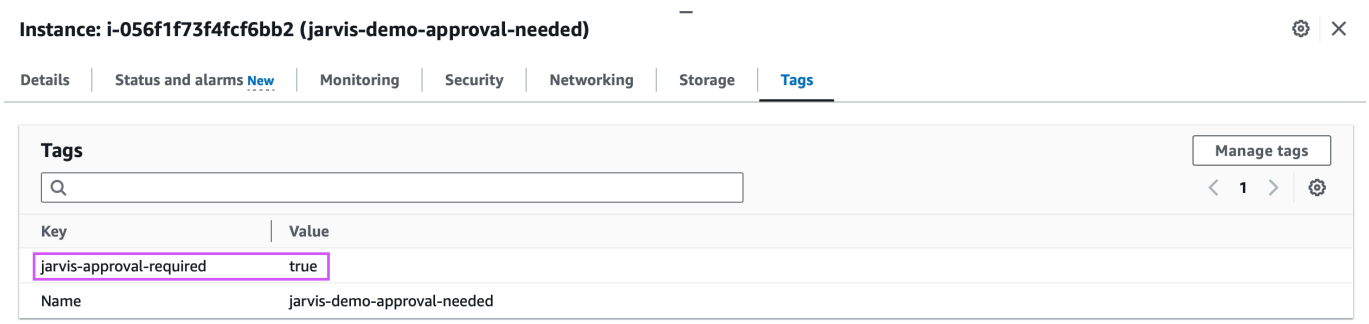

Figure 100. Resource in AWS tagged with "jarvis-approval-required:true".

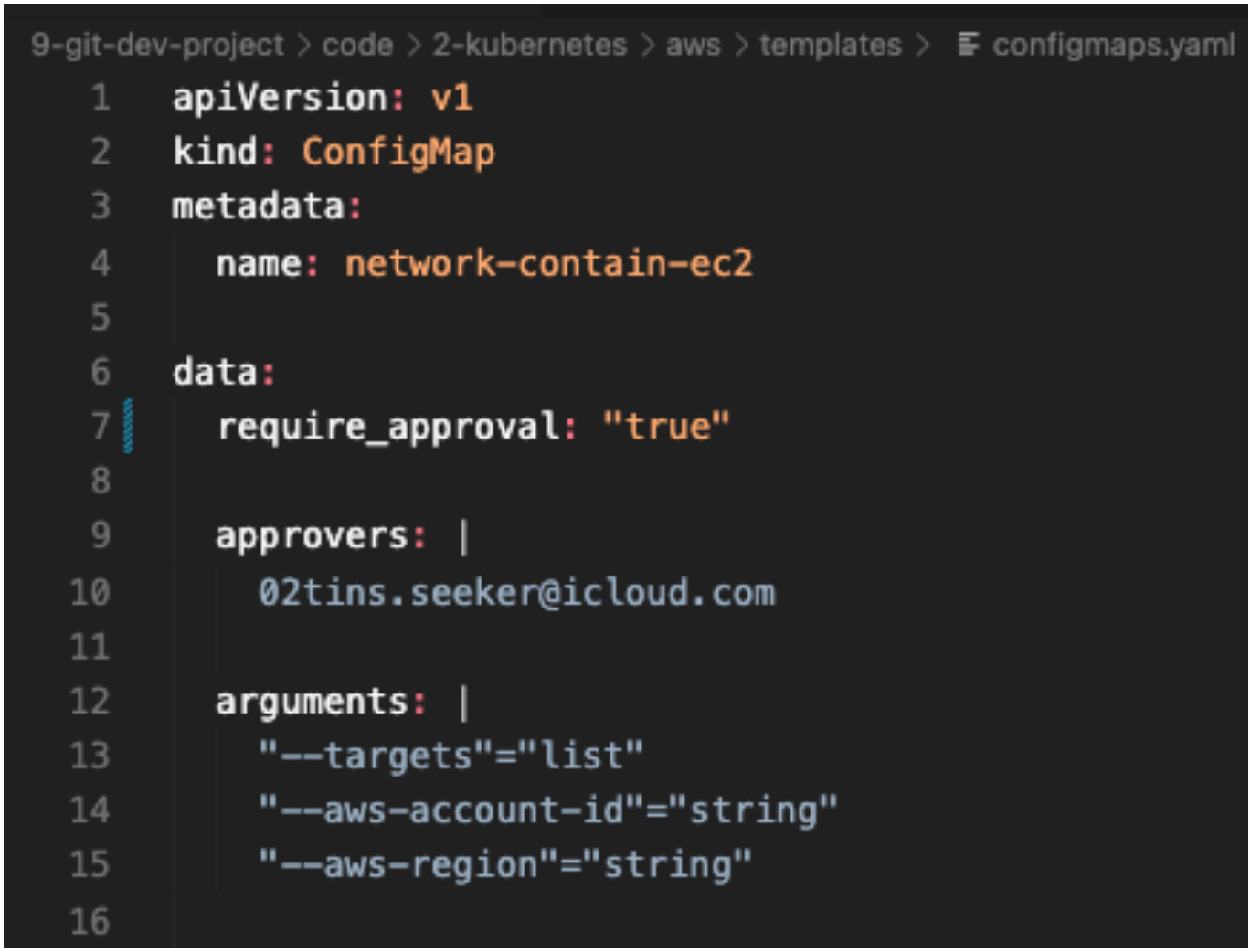

Figure 101. Automation service "network-contain-ec2"'s "require_approval" value has been set to true within its Configmap on line 7.

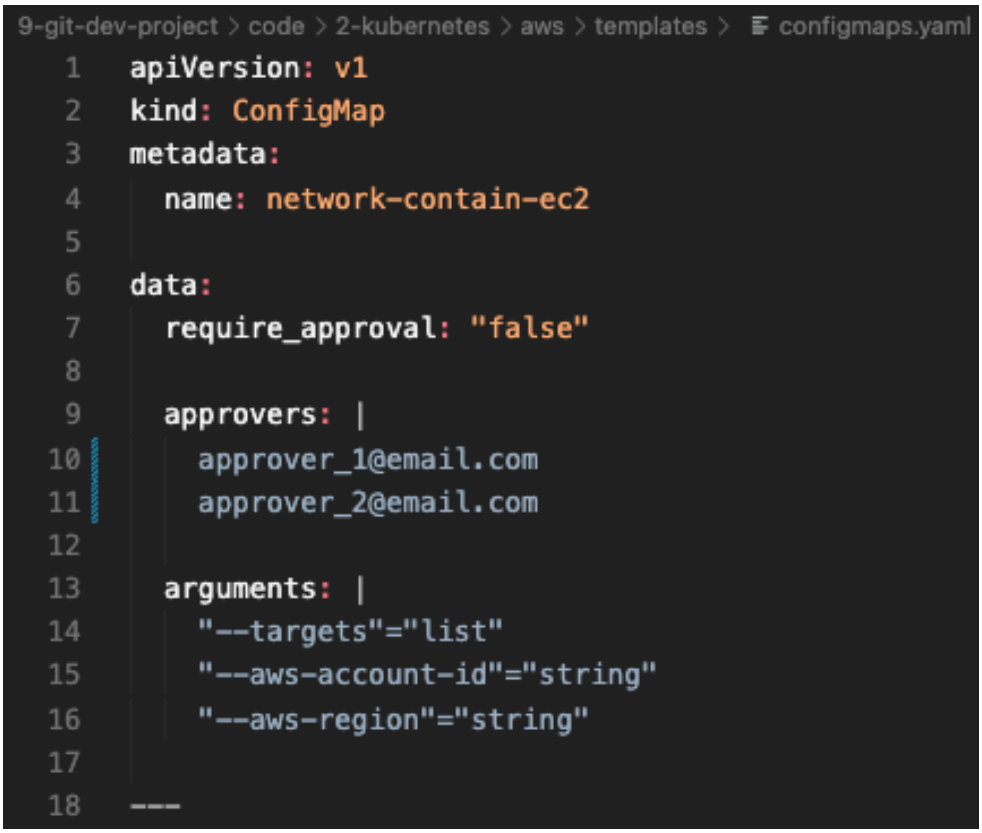

Figure 102. You will need to update the approvers list within each Configmap located in the AWS helm chart. This list should include the Slack user's email addresses that are allowed to approve actions for the particular service. In this figure's case, Slack users "approver_1@email.com" and "approver_2@email.com" will be authorised to approve actions using the "network-contain-ec2" service. These changes should be pushed to your GitHub repository to take affect.

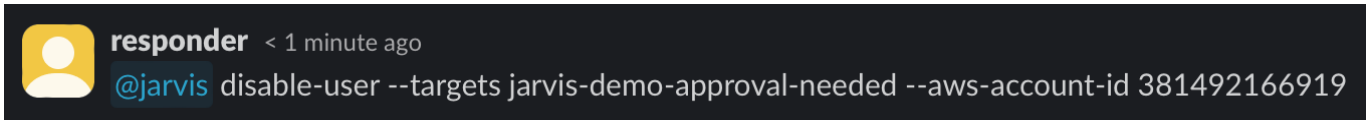

Figure 103. Responser user prompts automation to disable an IAM user.

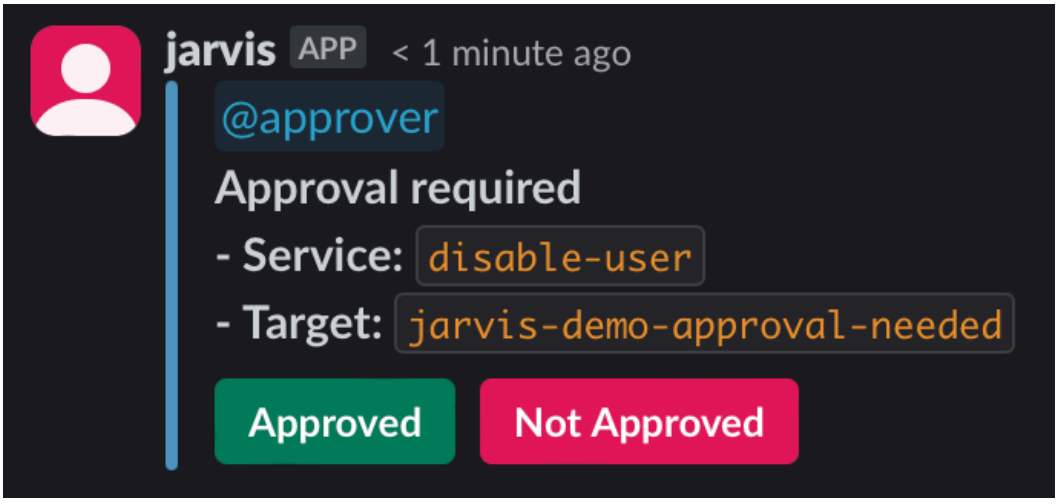

Figure 104. Automation responds with approval prompt for action. The message mentions the list of authorised approval users along with the service requested and target resource. The approver can respond by clicking either "Approved" or "Not Approved"

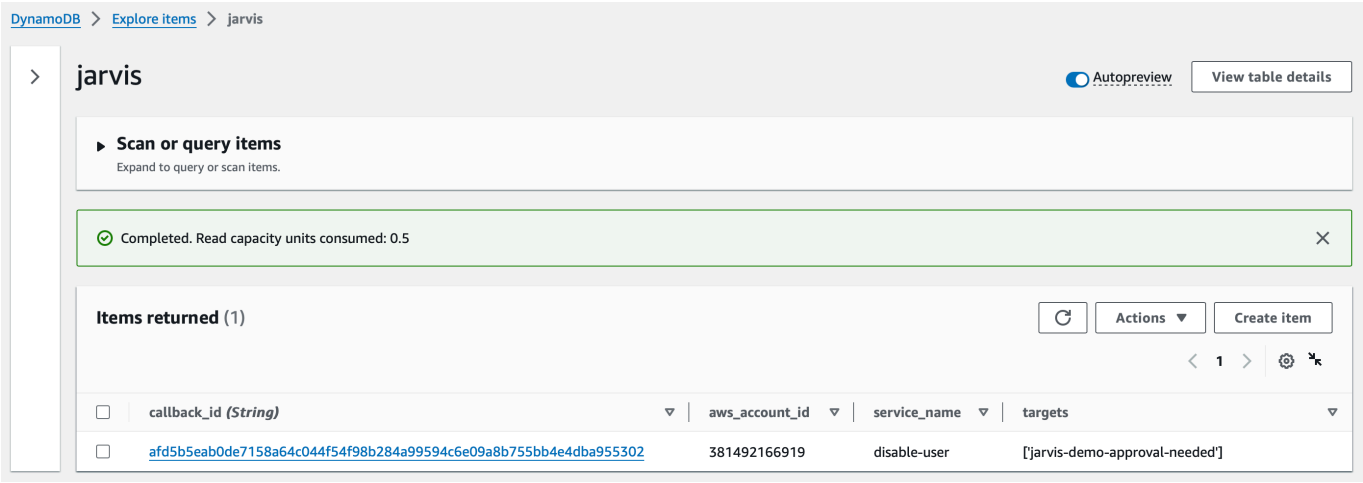

Figure 105. Approval request is stored in DynamoDB table. This allows for the Kubernetes resources to remain ephemeral but still facilitate asynchronous approval responses from users.

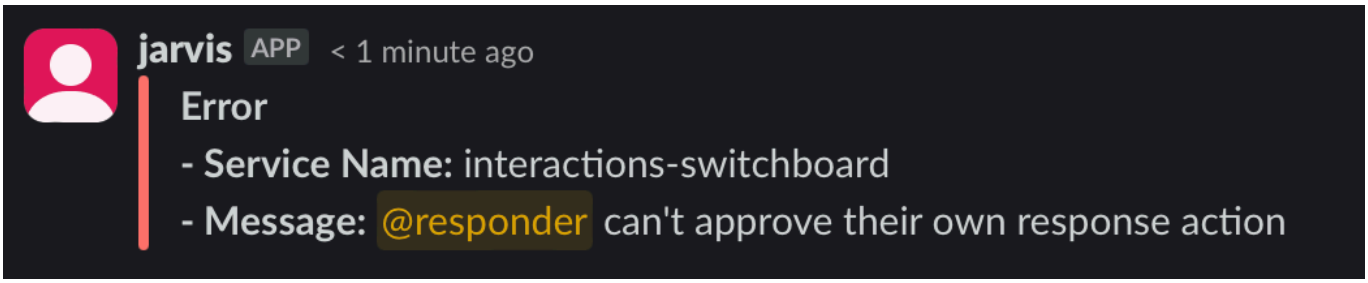

Figure 106. The responder user (original author of prompt seen in Figure 103) tries to approve their own request. This has been denied as self-approval is not permitted with an error message sent.

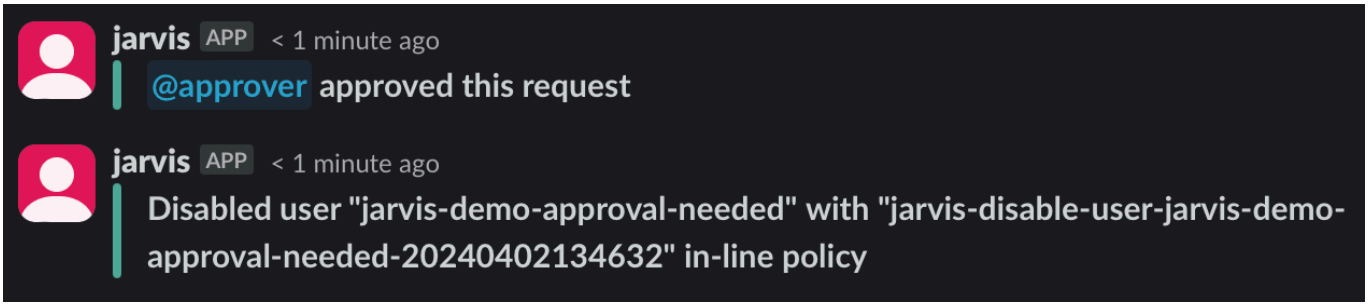

Figure 107. The approver user has clicked "Approved" on the message shown in Figure 103. This is acknowledged and the action is completed.

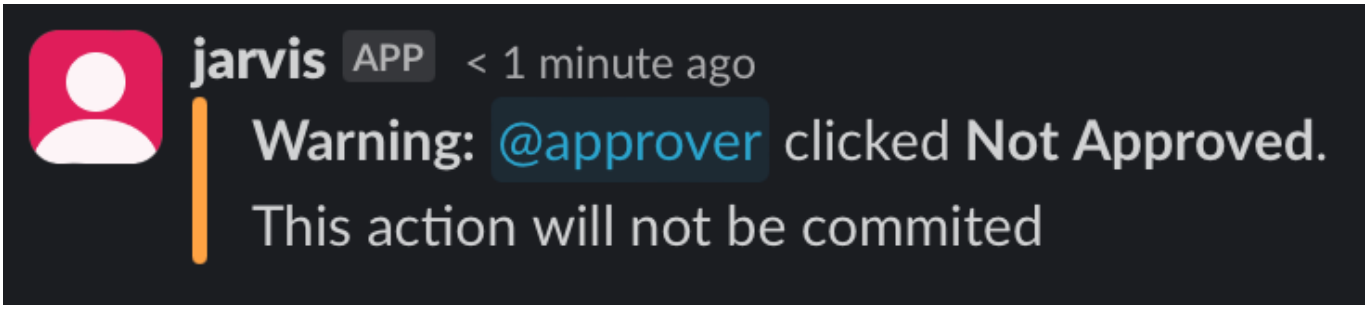

Figure 108. The approver user could have also click "Not Approved" which would result in a warning message to both acknowledge the response but also inform the original author that the action won't be completed.

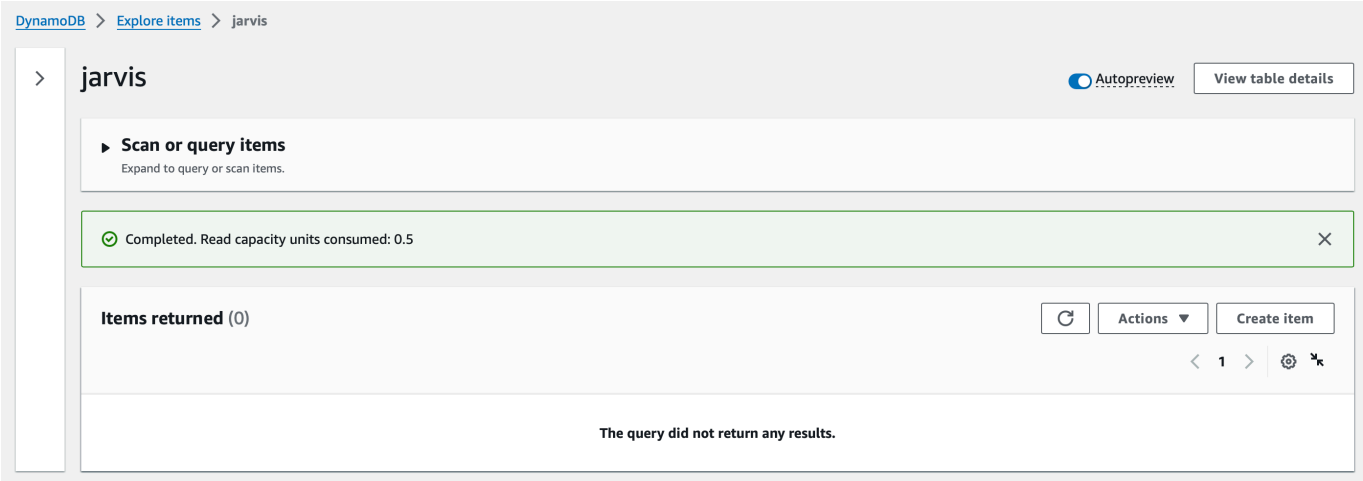

Figure 109. The approval request is deleted from DynamoDB as it is no longer needed after a authorised response is received.

Glossary

Term	Explanation
AMI (Amazon Machine Image)	A resource within AWS that contains the basic software for running an EC2 instance.
API (Application Programming Interface)	Allows for two different devices to communicate through a standard communicate language.
Application	Software that performs a function
Artificial Intelligence	The measurement of computer's intelligence including their ability to perform automated tasks and learn from sample material.
Authentication	Providing proof that one's self is who they say they are.
Automation	Using a computer to perform a task independently, but possibly triggered from a human action.
AWS (Amazon Web Services)	Amazon's CSP
AWS Account (Amazon Web Services)	A resource within AWS the contains other resources.
AWS Management Console (Amazon Web Services)	The website used to interact with an AWS account.
AWS Role (Amazon Web Services)	Identity in AWS that is assumed by a user or resource for a period of time when completing a specific function.
Bash	A programming language used to interact with the Unix / Linux shell.
Cloud Native	A system that is designed to run within a CSP.
Codebase	Collection of written software that forms an application
Containers	Is a way of packaging software and all its dependencies into a single entity which can be ran easily.
CSP (Cloud Service Provider)	An organisation that provides remote access to computing resources in exchange for a fee.
Data	Information
Dependency Manager	An application that installs and manages required software for others to run.
Dockerfiles	A set of instructions that define what software should be packaged into a container.
DynamoDB	A service offered by AWS which stores data. Each row of data has a primary key which is used to distinguish it from other rows.
EC2 (Elastic Compute Cloud)	A service offered by AWS. It provides customers the ability to rent access to computers for running workloads.
EKS (Elastic Kubernetes Service)	A service offered by AWS that creates a Kubernetes cluster.

Term	Explanation
Encryption	Applying mathematical logic to data so it can only be read by specific, authorised entities.
Environment Variables	A concept in computer science in which a value is assigned to a unique key so it can be quickly referenced.
Feedback Loop	A practice in software development which aims to quickly learn from mistakes by re-working a plan from continuous testing and user feedback.
GitHub	Online service that is used to store and version computer code.
GUI (Graphical User Interface)	Allows users to interact with computer systems via icons. An example of a GUI would be a website.
Hashing	A computational process applied to some data to produce a unique, fixed length output. Proper hashing algorithms' outputs should not be reversible and should not produce the same result for different input data.
HTTP (Hyper Text Transfer Protocol)	A set of rules that define how computers can send and receive network traffic.
IaC (Infrastructure as Code)	A method of configuring compute resources, usually within a CSP, via code rather than manually clicking buttons on a website.
IAM (Identity and Access Management)	A service offered by AWS that facilitates permissions and identities.
Idempotent	Value doesn't change when operated on
Incident Response	In terms of Cyber Security, it is the practice of responding to active intrusions or events that are likely to result in active intrusions.
Kernel	Software that manages the core functions of a computer.
Kubectl	An application used to interact with Kubernetes.
Kubernetes	Software that manages containers
Kubernetes Data Plane	Group of worker nodes
Kubernets Configmap	A resource within Kubernetes that stores configuration values. This data can then be included into a container either through environment variables or files.
LLM (Large Language Model)	A form of AI that can have general understanding of human language. An example would be OpenAI's ChatGPT.
Load Balancer	A resource within AWS that routes network traffic to a group of systems, distributing the traffic evenly to avoid over-burdening an individual system.
Logical Security Controls	Technologies put in-place to reduce the risk of an intrusion via virtual means. Some examples include anti-virus, multi-factor authentication and firewalls.

Term	Explanation
Micro-service	A concept in computer science in which a single function is programmed and provides support to other micro-services by sending and receiving network traffic.
Natural Language	Human language, either written or spoken.
Network Proxy	Sending data first to an intermediary for specific processing which then forwards to the intended destination.
OAuth (Open Authorisation)	Is a standard used to grant systems access to data on other systems without the need of a password.
Open Source	Material that is made available to the public free of charge
Pair Programming	A practice in software development in which two developers collaborate on code together. Usually, one will write the code and another provide feedback.
Persistent Data Storage	Maintaining data's integrity on a storage device, even after power loss.
Physical Security Controls	Technologies put in-place to reduce the risk of an intrusion via physical means. Some examples include door locks, CCTV and man-traps.
Python	A programming lanaguage
Playbook	A grouping of specific, individual automation steps to complete a larger task.
Port Binding	A concept in computer science that allows for a specific system to receive network traffic by assigning it a port number.
Primary / Hash Key	Assigned to a piece of data within a database. It is used to identify a specific row within a table and therefore needs to be unique.
Privilege Escalation	The act of increasing what one is allowed to do in a system.
Programming Language / Code	Collection of rules that define how computers can be instructed to conduct actions on a Human's behalf.
Programming Language Runtime	Software used to run the written code.
Script	Code written to perform a specific function.
SOAR (Security Orchestration, Automation and Response)	Technologies used to automate security response actions.
SaaS (Software as a Service)	A type of system that provides a service to users via a license and requires minimal setup.
SDLC (Software Development Life Cycle)	A practice in software development to organise the programming of a system with the aim of producing high quality systems.
Security Group	A resource within AWS that controls network access to EC2 instances.

Term	Explanation
Service Mesh	Dedicated infrastructure layer providing extension capabilities to existing software without altering said software's code.
SHA256	Is a hashing algorithm which produces a 256 bit (64 character output)
Slack	An application that provides communication capabilities, aimed at organisations.
Slack App / Bot	Is software that aims to automate actions within slack channels, usually triggered by user activity.
Snapshot	A service offered by AWS that copies an EC2 instance's data which can then be loaded onto another instance or used as a backup.
SQL (Structured Query Language)	A programming language used to manipulate databases.
Stateless	A concept in computer science in which a system doesn't permanently store data.
stdout (Standard Output)	A concept in computer science which provides a simple text output functionality to applications.
Syntax	The rules that defines a programming language, similar to grammar for natural language.
S3 (Simple Storage Service)	A service offered by AWS to store files on a remote storage drive.
Tag	Are configurable key-value pairs within AWS that allow users to assign custom data to resources. These can then be used to group, control access or measure costs for resources.
Threat Actor	A malicious individual or group who's aim is to compromise the security of a computer system through un-authorised manipulation.
TLS (Transport Layer Security)	A method of securing computer network traffic.
TLS Certificates (Transport Layer Security)	Method of validating the sender and / or receiver of data along with encrypting said data via cryptography.
URL (Uniform Resource Locator)	Is an address for a resource on a computer network. The internet uses these so users can find websites, for example www.google.com is a URL.
Virtual Machine	Logical separation of a physical computer's resources to form individual, virtual computers.
Worker Node	A computer that runs containers and forms a Kubernetes cluster.
YAML (Yet Another Markup Language)	Is a human-readable standardised language used to write files, usually for specifying the configuration of a system.