# National College of Ireland

## Project Submission Sheet

| | | | |
|---|---|---|---|
| **Student Name:** | EZEKIEL ADEDAYO AYANDA | | |
| **Student ID:** | 23129522 | | |
| **Programme:** | MSc Cloud Computing | **Year:** | 2024 |
| **Module:** | Configuration Manual | | |
| **Lecturer:** | Dr. Rashid Mijumbi | | |
| **Submission Due Date:** | Wednesday, 29th January 2025 | | |
| **Project Title:** | Scalability Optimization in Identity Management Systems for Cloud-Native Applications | | |
| **Word Count:** | …… | | |

**I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.**
**ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.**

| | |
|---|---|
| **Signature:** | Ezekiel Ayanda |
| **Date:** | 27/01/2024 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS:**

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

National College of Ireland

# Configuration Manual

MSc Research Project
Cloud Computing


Ezekiel Ayanda
Student ID: 23129522

School of Computing
National College of Ireland


Supervisor: Rashid Mijumbi

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Ezekiel Ayanda |
| **Student ID:** | 23129522 |
| **Programme:** | Cloud Computing |
| **Year:** | 2024 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Rashid Mijumbi |
| **Submission Due Date:** | 12/12/2024 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 1835 |
| **Page Count:** | 22 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Ezekiel Ayanda |
| **Date:** | 10th December 2024 |

## PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Ezekiel Ayanda
23129522

## 1 Introduction

This document serves as a comprehensive guide to perform a successful reproduction and configuration of the tools and technology utilized in the course of implementing this project with reference to the code artifact and online documentation were necessary. Considerate effort has been put into calling out possible issues encountered with setup and remediation steps taken to easy next reproduction of steps.

## Contents



## 2 Setup Prerequisites

For the scope of this project, certain versions of tools and platforms have been utilized. While it might not be mandatory, it is recommended that the same versions are used were necessary for best results and to avoid configuration mishaps.

Table 1: Azure Kubernetes Cluster

| **Component** | **Type/Version** |
|---|---|
| Compute Instance | Virtual Machine Scale Sets |
| Operating System | Ubuntu Linux |
| Orchestration Tool | Kubernetes (Version 1.29.9) |
| Container Tool | ContainerD (Version 1.6.26) |

Table 2: VMSS / Node Specification

| | |
|---|---|
| vCPU | 2 vcores |
| Memory | 7 GB |
| Node VM Size | $\text{Standard}_{D}S2_{v}2$ |
| OS Type | Ubuntu Linux |
| Cost of services | $0.146/Hour |

# 3 Azure Kubernetes Cluster

As a prerequisite to all configurations in this project, it is expected that you already have at least contributor access to the Azure Kubernetes cluster.

## 3.1 Install AZ CLI

Azure CLI is available for download and installation for any of your preferred environments (Windows, macOS and Linux environments). This can be accessed at Azure CLI Installation Guide

## 3.2 Deploy AKS Cluster

**Sign into Azure CLI** Use the following command to sign into your Azure account:

```
az login
```

**Set Subscription Context** Set the subscription context to your subscription:

```
az account set --subscription <sub name or id>
```

**Create AKS Cluster** Run the following command to create the AKS cluster:

```
az aks create \
--resource-group <resource-group-name> \
--name <aks-cluster-name> \
--node-count <number-of-nodes> \
--node-vm-size <vm-size> \
--generate-ssh-keys \
--kubernetes-version 1.29.9 \
--network-plugin azure
```

## 3.3 Install Kubectl, Docker and Helm Chart

Most of the deployments in the cluster are deployed using Helm as the package manager. Docker is needed for build the Java app and Kubectl is required to run commands against cluster. Official guides for installation of these can be followed for installation based on Operating system.

- **Kubectl**: https://kubernetes.io/docs/tasks/tools/
- **Docker**: https://docs.docker.com/engine/install/

- **Helm**: https://helm.sh/docs/intro/install/

### 3.4 Nodes and Namespace configuration

Create dedicated nodes for different parts of the cluster along with corresponding namespaces. Three user node pools can be created alongside three namespaces. The system node pool is specifically reserved for native Kubernetes operational workloads.

Table 3: Cluster workload Recommended Mappings

| Node pool | Namespace |
|---|---|
| appnode1 | appnode |
| idmsnode1 | idmsnode |
| monitoring | monitoring |

Defining scoped nodes and namespaces ensures proper grouping of workloads within the cluster, leveraging node selectors and namespace parameters for deployment management. User nodes can be created via kubectl or via the Azure portal using the system specifications in 2

## 4 Java SpringBoot Application

### 4.1 Integration and setup

Configuration prerequisites include:

- Realm Name
- Client ID
- Client Secret

Include the necessary dependencies for `OAuth2` and Keycloak in your `pom.xml`:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-spring-boot-starter</artifactId>
    <version>26.0.6</version>
</dependency>
```

Create a security configuration class to enable OAuth2-based authentication. This handles the connectivity to Keycloak and redirects users for login in `/Java Springboot App/src/main/java/co`

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
       http) throws Exception {
        http
            .cors(AbstractHttpConfigurer::disable)
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(req -> req
                .requestMatchers(AntPathRequestMatcher.antMatcher
                   ("/**")).permitAll()
                .anyRequest()
                .authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.
                   STATELESS));

        return http.build();
    }

}
```

## 4.2 Push App to Docker or Azure Container Registry

Build the Docker image

```
docker build -t <your-dockerhub-username>/java-spring-app:v1 .
```

Push the Docker image to a container registry

```
docker push <your-dockerhub-username>/java-spring-app:v1
```

Creating the Application in the Kubernetes Cluster

The Deployment manifest file can be found in `/yamls/java-app-deployment.yml`. Update your docker image and selector. Also set Environment variable values to match Keycloak deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-app
  labels:
    app: java-app
spec:
  replicas: 2
  selector:
```

```
    matchLabels:
      app: java-app
  template:
    metadata:
      labels:
        app: java-app
    spec:
      containers:
      - name: java-app
        image: <your-dockerhub-username>/java-spring-app:v1
        ports:
        - containerPort: 8080
        env:
        - name: iam.integration.keycloak.url
          value: "http://<keycloak-host>/"
        - name: iam.integration.keycloak.client-id
          value: "<ClientID>"
        - name: iam.integration.keycloak.realm
          value: "<realm name>"
        - name: iam.integration.keycloak.username
          value: "admin"
        - name: iam.integration.keycloak.password
          value: "adminpassword"
        - name: iam.integration.keycloak.client-secret
          value: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

The Service manifest file can be found in `/yamls/java-app-deployment.yml`

```
apiVersion: v1
kind: Service
metadata:
  name: java-app-service
spec:
  selector:
    app: java-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

Confirm Java App Pod and Service in AKS

```
kubectl get pods -n appnode
kubectl get svc -n appnode
```

# 5 Identity Management System

## 5.1 Keycloak Installation

This configuration should get Keycloak up and running for a development use case. A full list of Bitnami Helm chart parameters for Keycloak can be found on **Bitnami's GitHub repository**.

```
helm install keycloak bitnami/keycloak \
--namespace idmsnode \
--set service.type=LoadBalancer \
--set auth.adminUser=admin \
--set auth.adminPassword=<securepassword> \
--set nodeSelector.agentpool=idmsnode1 \
--set replicaCount=3 \
--set persistence.enabled=true \
--set persistence.size=10Gi \
--set postgresql.enabled=true \
--set postgresql.postgresqlUsername=admin \
--set postgresql.postgresqlPassword=<securepassword> \
--set postgresql.postgresqlDatabase=keycloak \
--set resources.requests.cpu=500m \
--set resources.requests.memory=1Gi \
--set resources.limits.cpu=1 \
--set resources.limits.memory=2Gi
```

Confirm the pods, Service and Persistent Volume Claim deployment in AKS

```
kubectl get pods -n idmsnode
kubectl get svc -n idmsnode
kubectl get pvc -n idmsnode
```

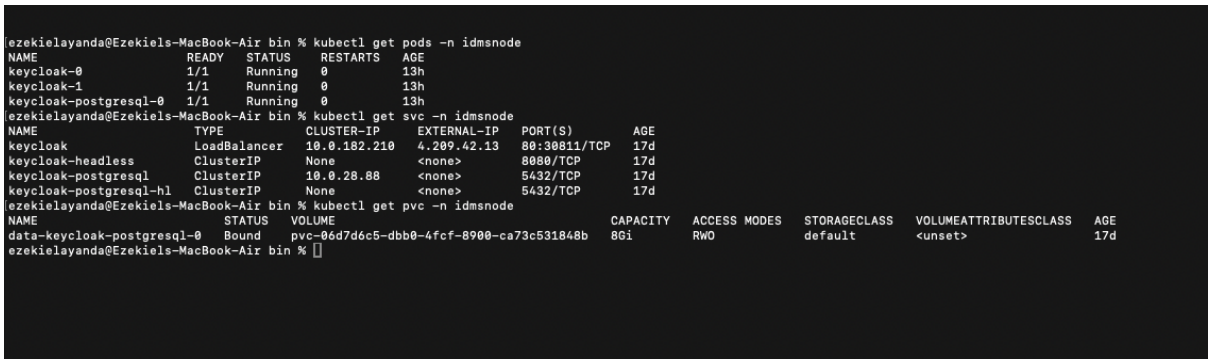


Figure 1: Keycloak pods, services and Persistent Data Volume for Postgresql

Obtain and verify Keycloak admin password

```
kubectl get secret keycloak -n idmsnode -o jsonpath="{.data.admin-password}"
| base64 --decode
```

Keycloak Admin console can be accessed over the internet with the External IP or via port Forwarding.

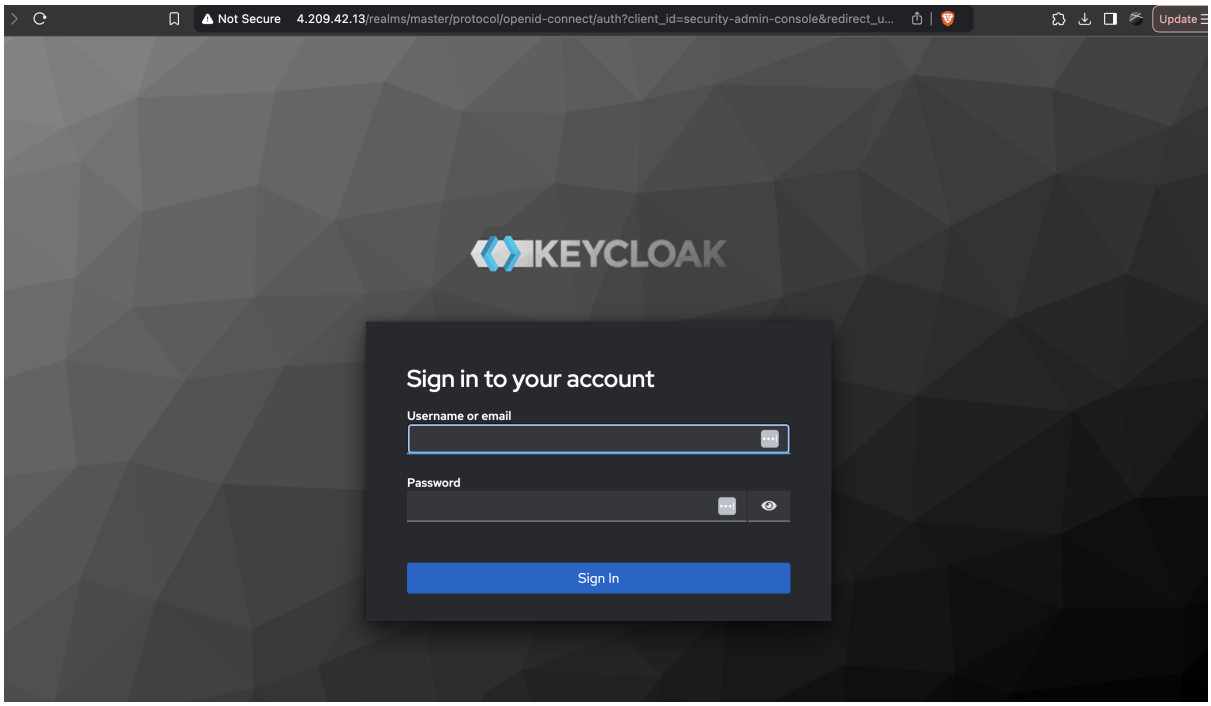


Figure 2: Keycloak Admin Console Login page

Enable Autoscaler for Keycloak which will be working with Infinspan to on-boarding new instances into the cache cluster. (Minimum replicacount to enable Keycloak clustering is 2). Update this in the `value.yaml` file

```
autoscaling:
enabled:  true
minReplicas:  2
maxReplicas:  3
targetCPU: 80
targetMemory:  80
```

Run `helm update` to update helm deployment with new parameters

```
helm upgrade keycloak bitnami/keycloak -f values.yaml -n idmsnode
```

## 5.2 Clustering configuration

To enable Keycloak for clustering, distributed cache management has to be enabled and Node discovery set.

Configure Distributed Caching with Infinispan by enabling caching and cache stack in values.yaml:

```
cache:
enabled:  true
stackFile:  ""
stackName:  kubernetes
```

```
extraEnv:
keycloak:
- name:  KC_CACHE
value:  kubernetes
- name:  KC_CACHE_STACK
value:  ispn
```

This can also be added into the keycloak-env-vars config file

```
kubectl edit configmap keycloak-env-vars -n idmsnode
```

```
KEYCLOAK_CACHE_STACK: kubernetes
KEYCLOAK_CACHE_TYPE: ispn
```

Node Discovery by Enabling JGroups

Add to `values.yaml` to configure node discovery. (`JGROUPS_BIND_PORT` range can be flexible)

```
- name:  JGROUPS_BIND_PORT
value:  "17900-18000"
- name:  JGROUPS_BIND_ADDR
valueFrom:
fieldRef:
fieldPath:  status.podIP
- name:  JGROUPS_STACK
value:  "kubernetes"
```

With a successful configuration, clustering activity can be verified through the logs of any instance, preferably the first one.

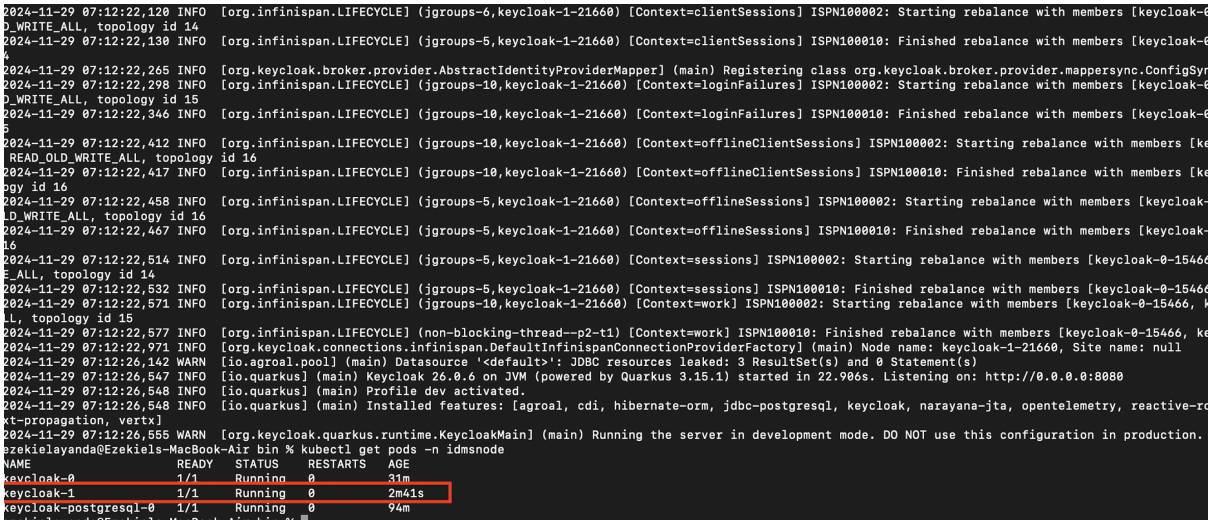


Figure 3: Cluster info from Infinispan in Keycloak pod logs

## 5.3 JavaScript Provider configuration

Ability to build custom scripts require the scriptsv2 feature, which is part of the preview feature. This needs to be added to the environment variable of Keycloak and initiated upon pod startup.

Edit ConfigMap

```
kubectl edit configmap keycloak-env-vars -n idmsnode
```

Add in either `--features=scripts` or `---features=scripts` and save ConfigMap

The `values.yaml` file is also updated with preview and SPI enabling.

```
- name:  KC_SPI_POLICY_JS_ENABLED
value:  "true"
- name:  KC_FEATURES
value:  "preview"
```

Prepare the JavaScript policy and Meta in a folder following the structure

```
META-INF/keycloak-scripts.json
script-example-policy.js
```

Copy the folder from local machine to providers folder in the keycloak server

```
kubectl cp ./script-example-policy \
idmsnode/keycloak-0:/opt/bitnami/keycloak/providers
```

Connect into console of the Keycloak server

```
kubectl exec -it keycloak-0 -n idmsnode -- bash
```

Compress the folder to a `.jar` file

```
cd /opt/bitnami/keycloak/providers/
jar cvf script-example-policy.jar -C script-example-policy .
```

Initiate a Build for the .jar file

```
./opt/bitnami/keycloak/bin/kc.sh build
```

Restart Keycloak in Development mode and redirect port to avoid issues with utilized port

```
/opt/bitnami/keycloak/bin$ kc.sh start-dev --http-port=8082
```

Perform a port forwarding to access the Keycloak over the browser

```
kubectl port-forward pod/keycloak-0 8081:8081 -n idmsnode
```

Ensure to enable "Authorization" in client settings for policy settings to be visible.

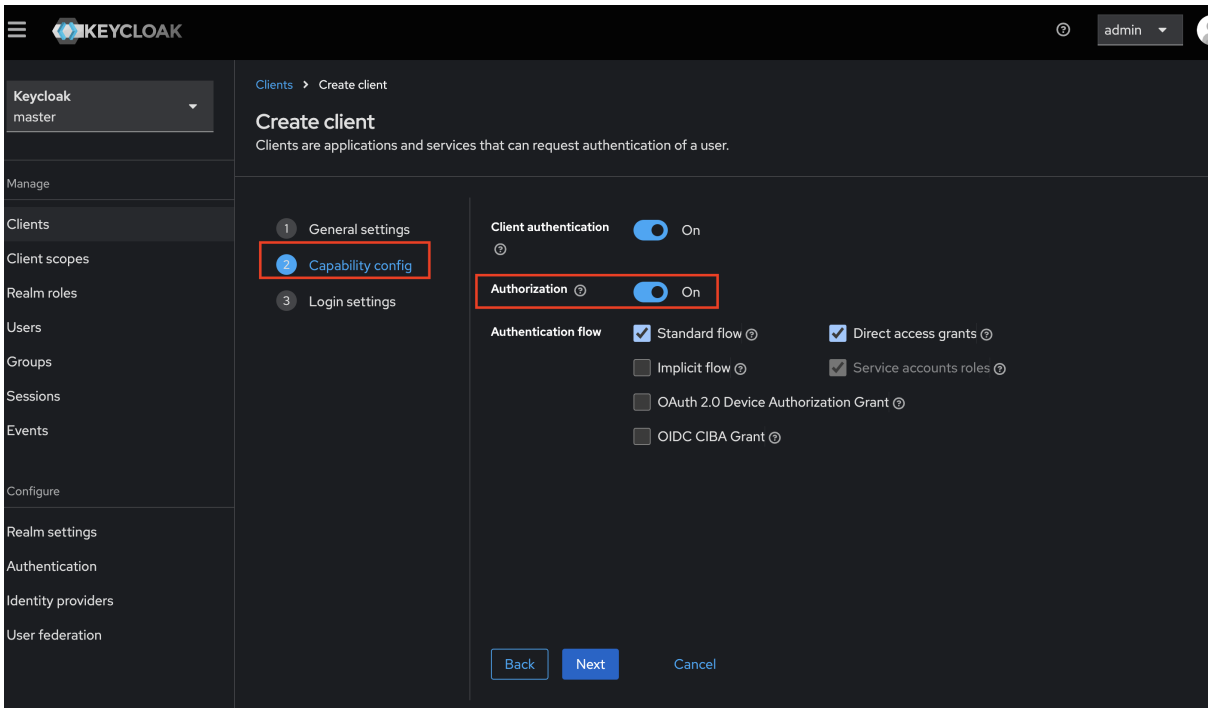


Figure 4: Authorization Toggle in Client Settings

Custom Policy will be accessible in policies under clients' configuration

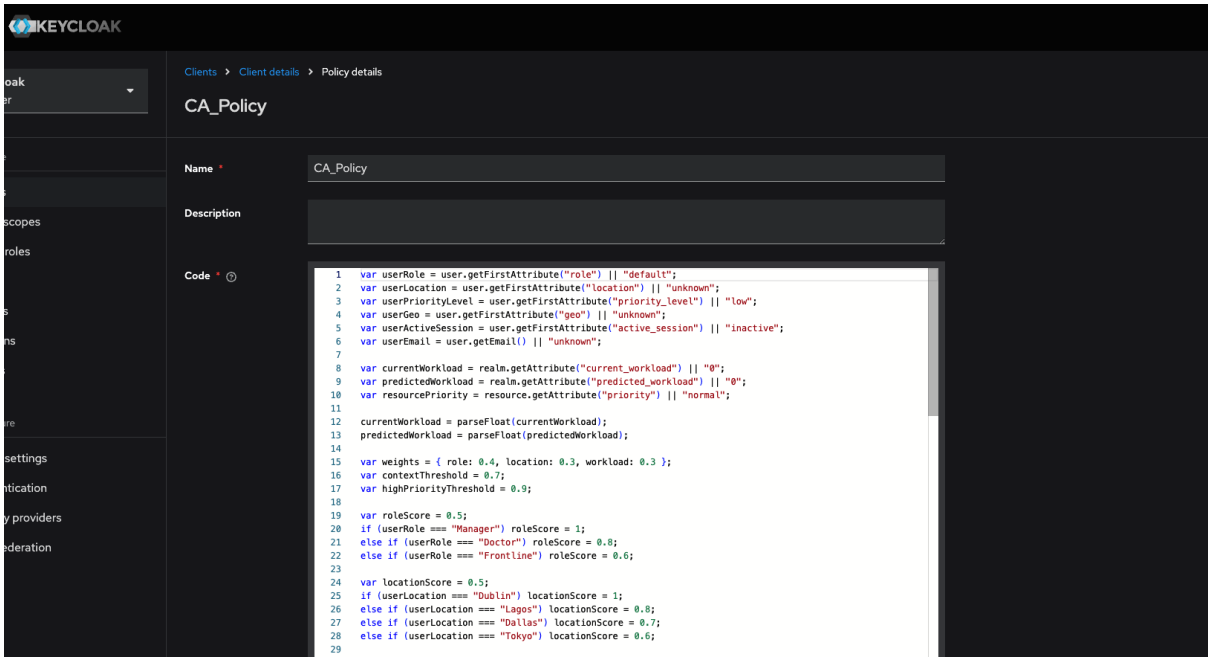


Figure 5: Custom Policy in Keycloak

# 6 Time Series Monitoring

## 6.1 Prometheus

In the terminal add the Helm repository for Prometheus and update repository

```
helm repo add prometheus-community \
https://prometheus-community.github.io/helm-charts
helm repo update
```

Install the downloaded chart to accurate node and namespace

```
helm install prometheus prometheus-community/prometheus \
--namespace monitoring \
--set nodeSelector.node=monitoring
```

Get the services in namespace and expose the Prometheus-server port

```
kubectl get svc -n monitoring
```

```
kubectl expose service prometheus-server \
--type=NodePort \
--target-port=9090 \
--name=prometheus-server-ext \
-n monitoring
```

Forward port to be accessible locally and access over the **Browser**: https://127.0.0.1:9090

```
kubectl port-forward svc/prometheus-server-ext 9090:80 -n monitoring
```

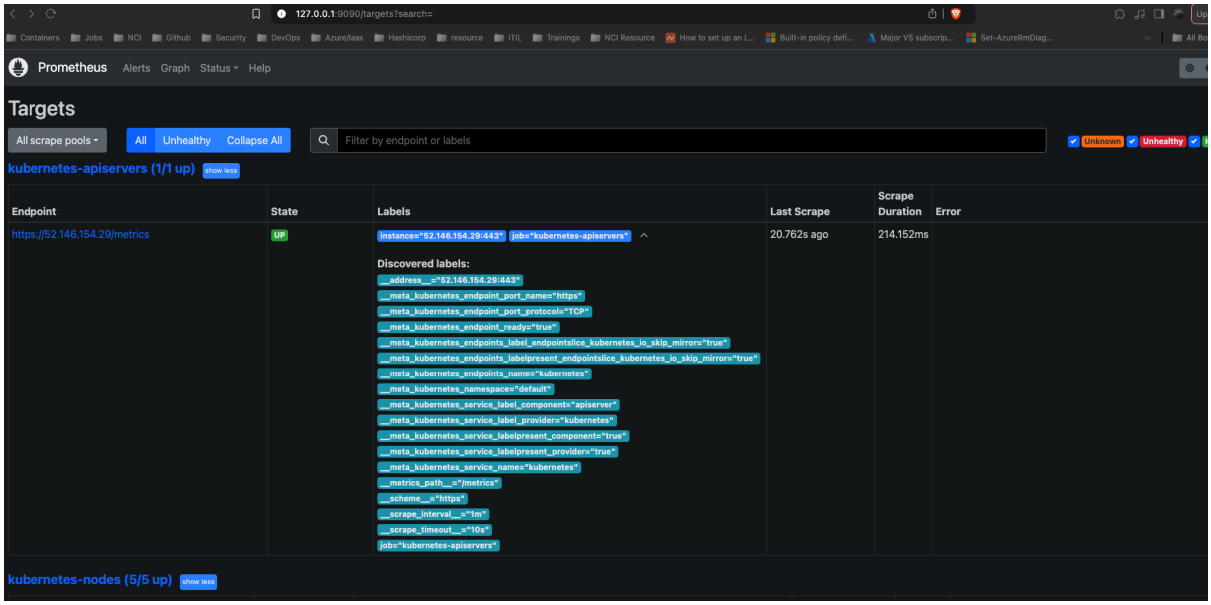


Figure 6: Prometheus Portal

Live metrics can be viewed at `https://<ipexposed:port>/metrics`.

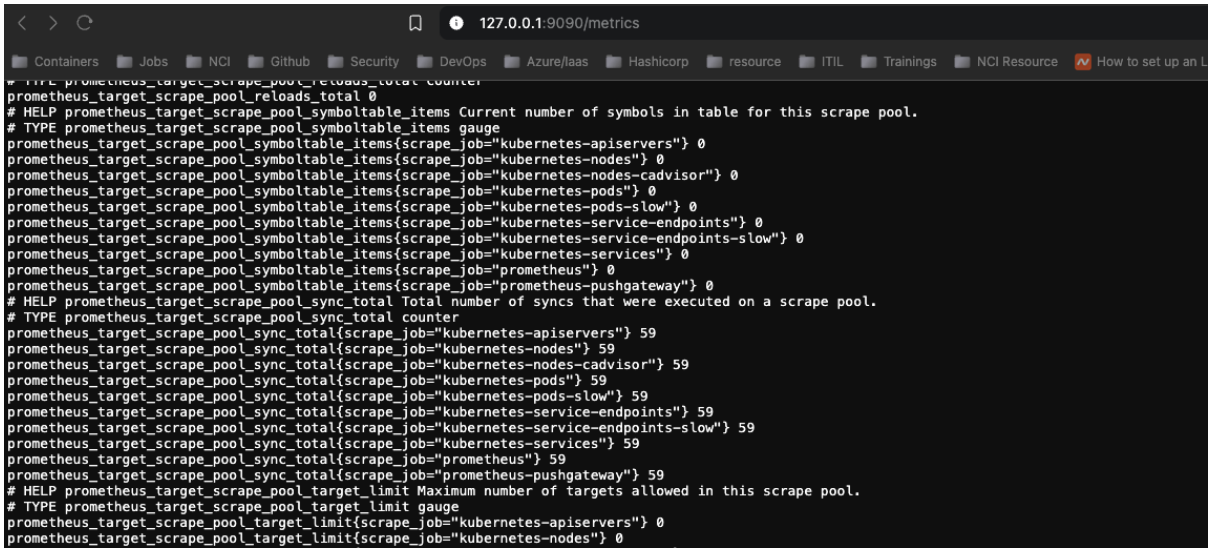


Figure 7: Prometheus Metrics

## 6.2 Grafana Dashboard

In the terminal add the Helm repository for Grafana and update repository

```
helm repo add prometheus-community https://grafana.github.io/helm-charts
helm repo update
```

Install the downloaded chart to accurate node and namespace

```
helm install grafana grafana/grafana \
--namespace monitoring \
--set nodeSelector.node=monitoring
```

Confirm running pod and service in the monitoring Namespace

```
kubectl get pods -n monitoring
kubectl get svc -n monitoring
```

Retrieve the admin password for Grafana login (Username is `admin`)

```
kubectl get secret grafana -n monitoring \
-o jsonpath="{.data.admin-password}" | base64 --decode
```

Forward port to be accessible locally and access over the browser at **Browser**: https://127.0.0.1:3000/login

```
kubectl port-forward svc/grafana 3000:80 -n monitoring
```

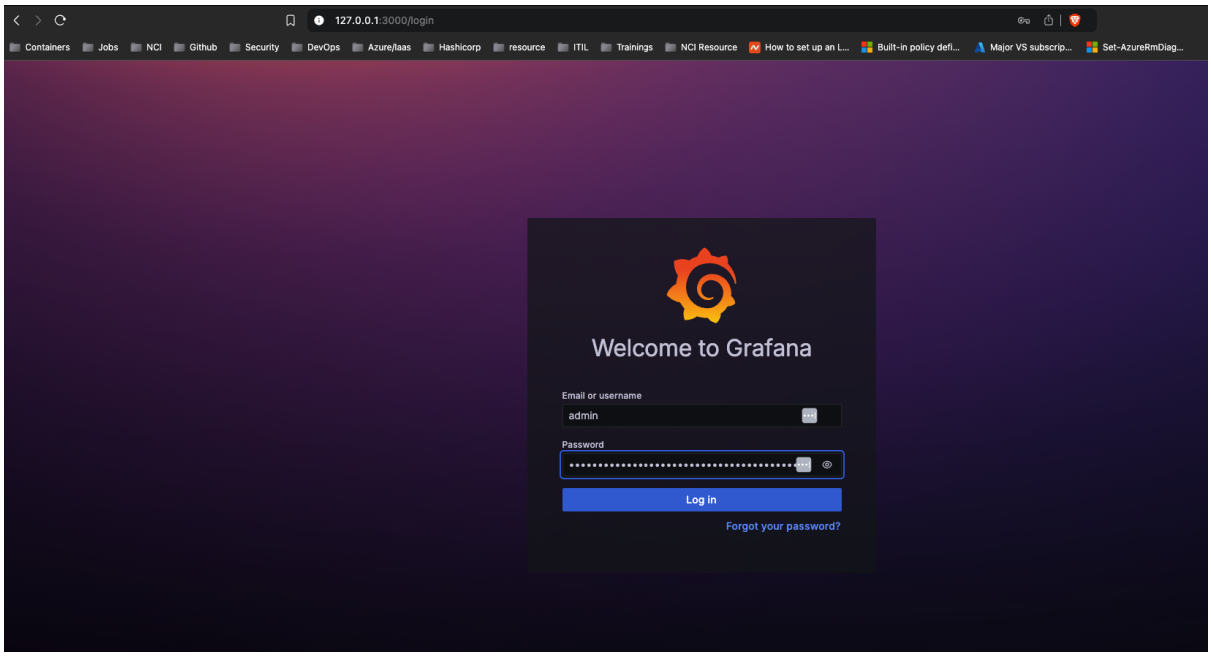


Figure 8: Grafana Login page

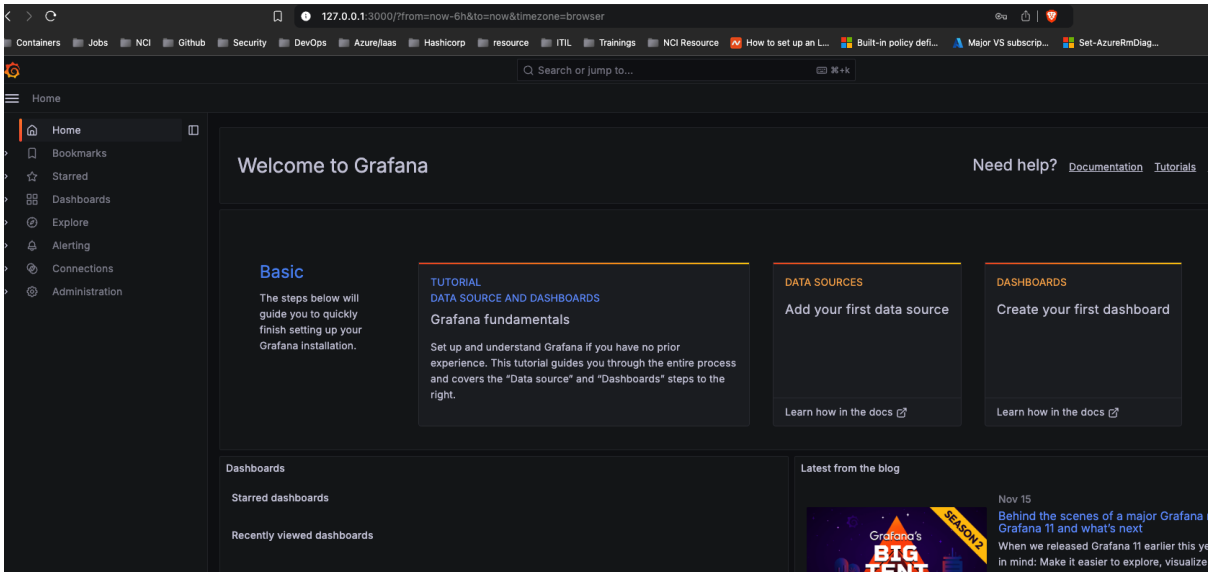


Figure 9: Grafana Console

To ensure persistence of data settings configured in Grafana, add a Persistent Volume Claim (PVC)

```
helm upgrade grafana grafana/grafana --namespace monitoring \
--set persistence.enabled=true \
--set persistence.size=10Gi \
--set persistence.storageClassName=default
```

Confirm presence of persistent volume claim

```
kubectl get pvc -n monitoring
```

Connect Grafana to Prometheus Data Source and add dashboard of choice based on desired metrics

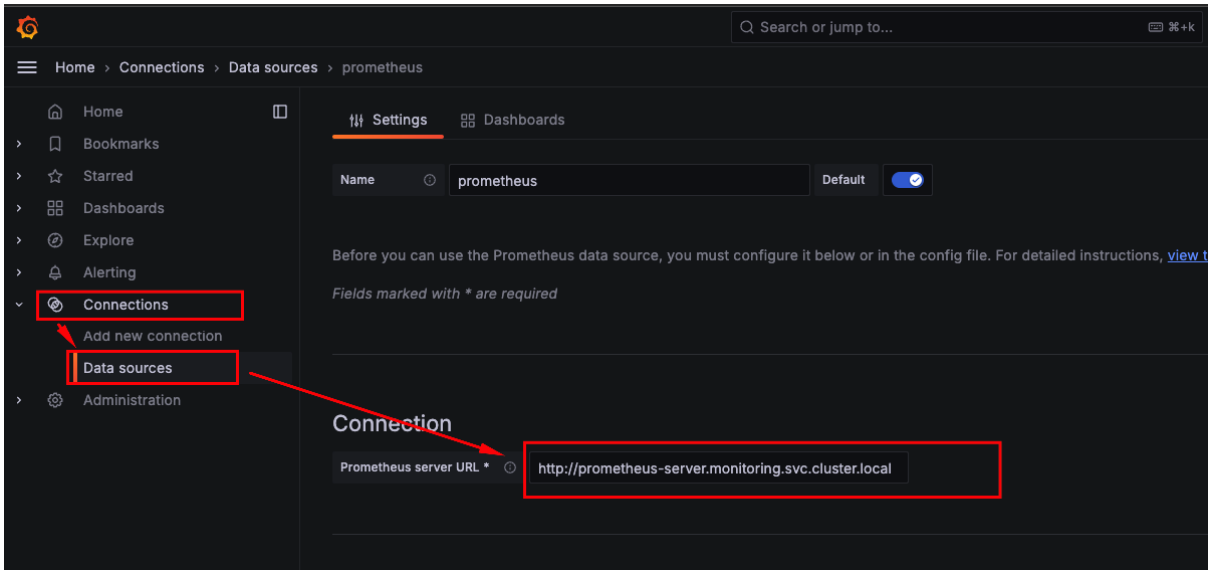


Figure 10: Adding Data Source

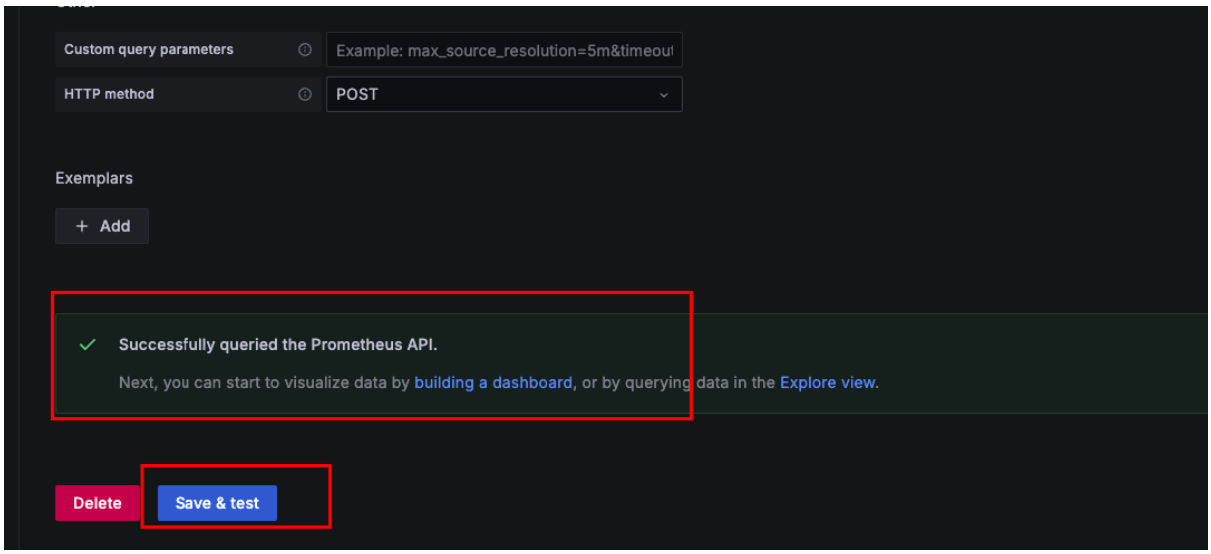


Figure 11: Saving Data Source


Figure 12: Grafana Dashboard

## 6.3 Queries for Metrics collection

These are the PromQL queries used to grab metrics furnished to the Grafana Dashboard.

CPU usage percentage for pods in a Namespace

```
sum(rate(container_cpu_usage_seconds_total{namespace="idmsnode"}[5m])) by
(pod) * 100
```

Memory Usage in Megabytes (MB) for all pods in a Namespace

```
container_memory_usage_bytes{namespace="idmsnode"} / 1048576
```

Http Requests total for pods in a Namespace

```
apiserver_request_total{resource="pods"}
```

Pod Additions into a Namespace

```
increase(kube_pod_status_phase{namespace="idmsnode", phase="Running"}[5m])
increase(kube_pod_status_phase{namespace="appnode", phase="Running"}[5m])
```

# 7 Load Testing

## 7.1 JMeter Configuration

Install JMeter on MAC OS.

**Note:** For other operating systems, download at https://jmeter.apache.org/download_jmeter.cgi.

```
brew install jmeter
```

Verify JMeter installation

```
jmeter -v
```


Figure 13: JMeter Installed

Launch Jmeter

```
jmeter
```

Load JMeter Test Plan file from downloaded artifact at `/Jmeter FIles/testplan1.jmx`

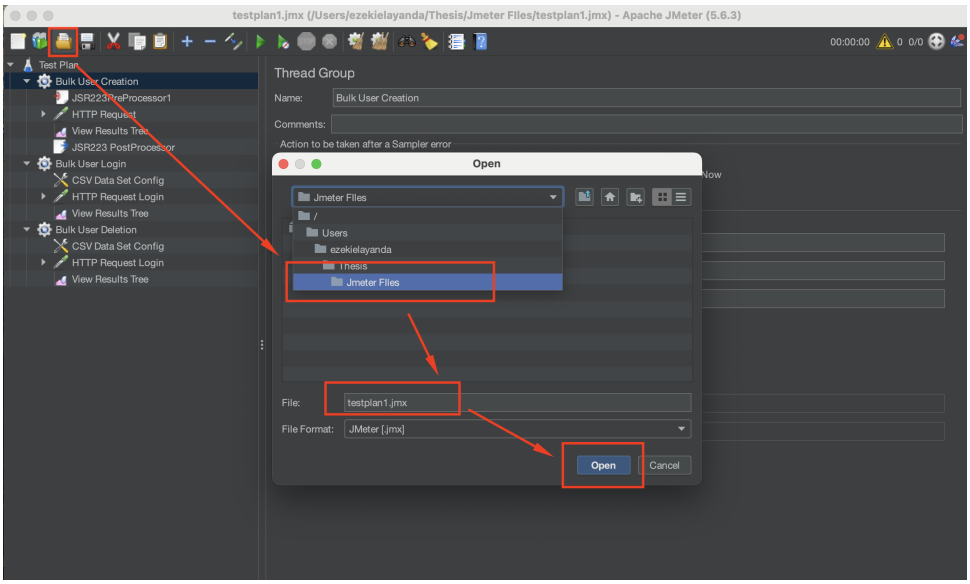


Figure 14: Load Test Plan

Three Thread Groups are present for User Registration, Login and Deletion respectively.

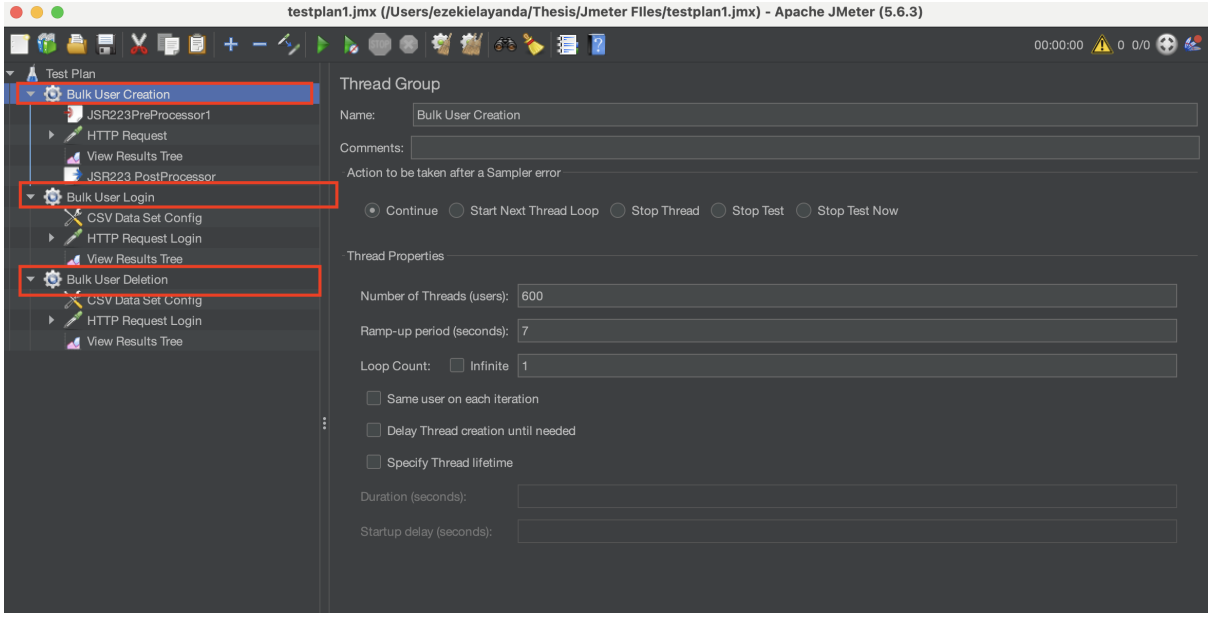


Figure 15: Thread Groups

Number of users to be created can be specified with ramp up period. The user creation request is forwarded from the Java application to Keycloak where users are created almost real time. Created users data are saved in `/path/userdata.csv`

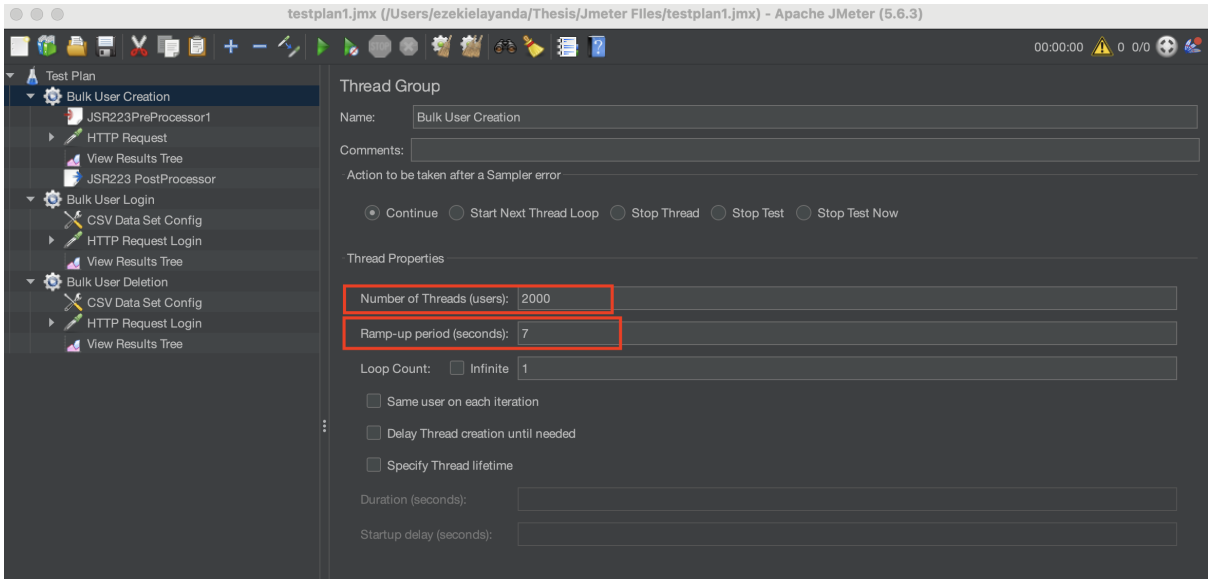


Figure 16: User Creation

User creation and data export to CSV are done at the JSR223 PreProcessor and Post-processor stages respectively utilizing groovy scripts saved in the artifact at:

```
/Jmeter FIles/usercreation.groovy
/Jmeter FIles/createduserexport.groovy
```

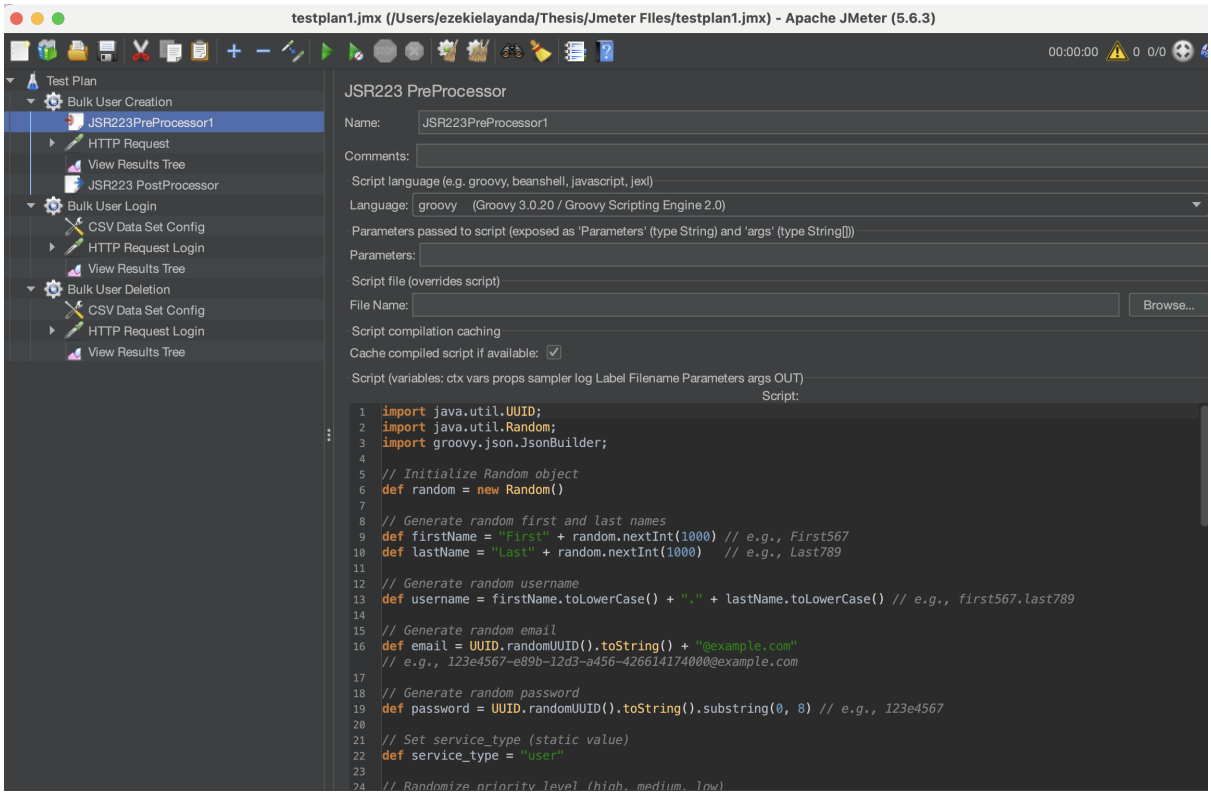


Figure 17: JSR223PreProcessor

## 7.2 (Optional) Run Login request on Azure Load testing

For more compute power, Azure load testing can be used to run the load testing due to it's more robust capacity. Load testing configuration in JMeter can be exported as as .JMX file and uploaded to Azure load testing.
You can create an Azure Load testing resource and create a new test in it by uploading the JMter script.

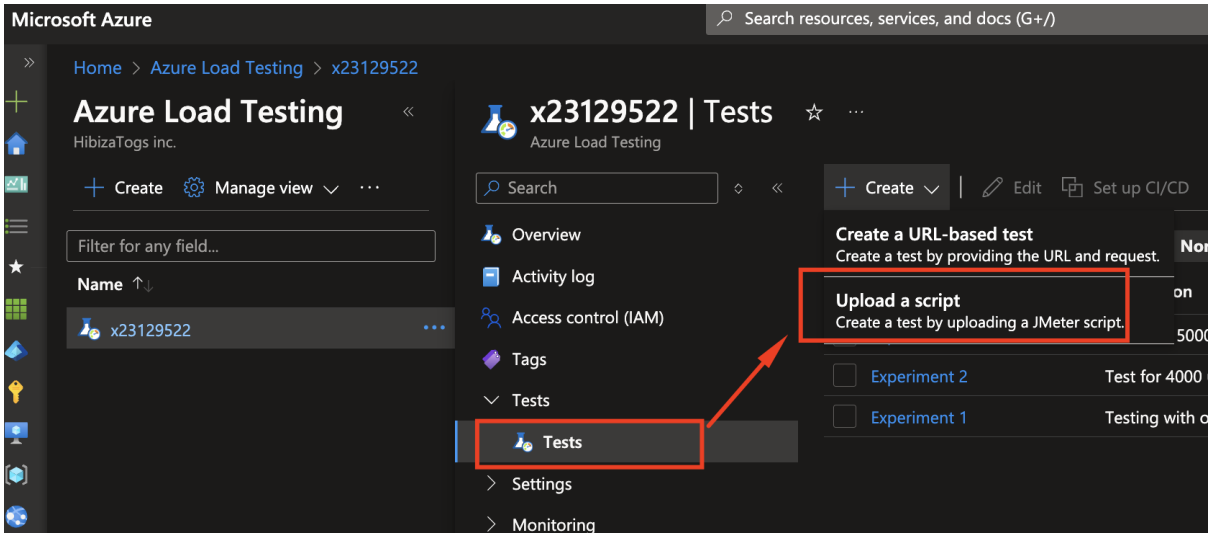


Figure 18: Create Test in Azure Load Testing

Upload the Jmeter file and the user data csv genrated for new users created in Keycloak

you can also choose to split the CSVs evenly between engines if you are using more than one engine (Completely optional).

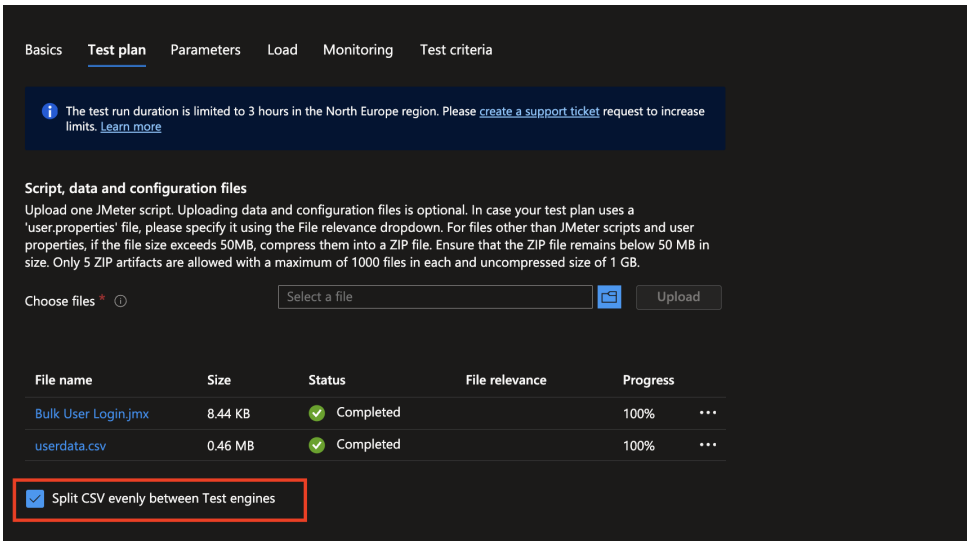


Figure 19: Upload Test plan

If you have a large csv file, you can split the workload on more than one engine instance and also generate the load from different regions. As this configuration has public endpoints enabled, we have selected "Public" for the traffic mode.

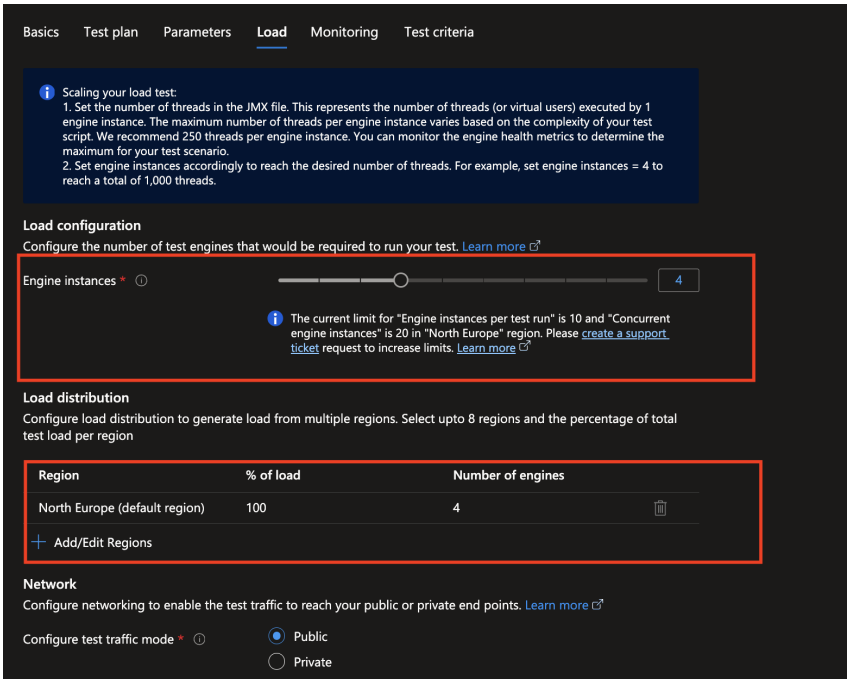


Figure 20: Load Configuration

Other settings can be skipped and then save configuration. Launch the test plan

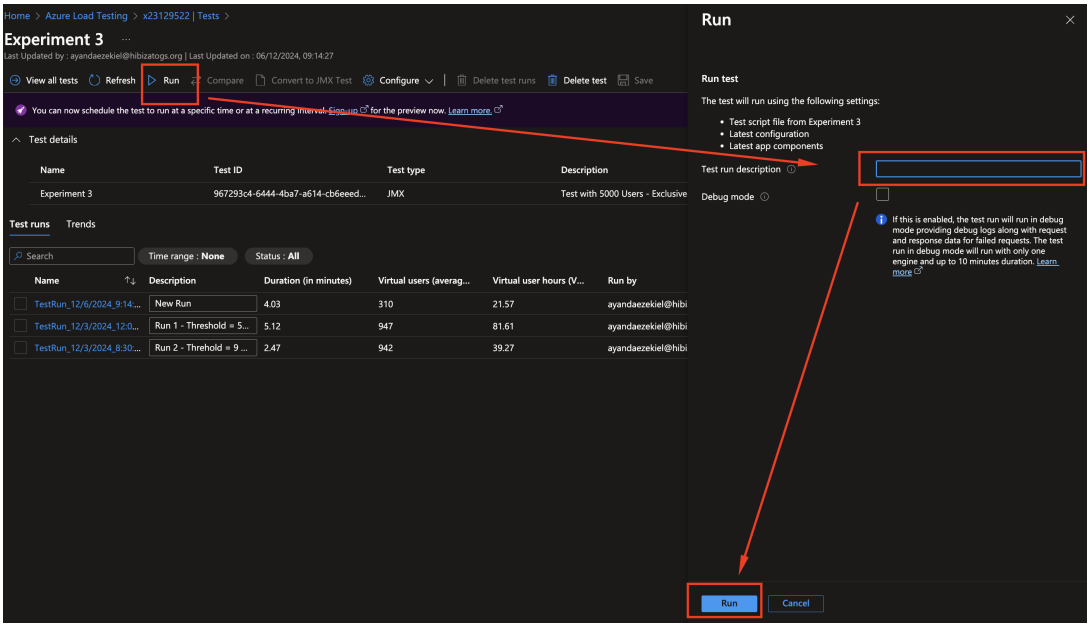


Figure 21: Test Plan Launch

## 7.3 (Optional) Add Backend Listener for Azure Application Insights

Backend Listeners can be added to the test plan in other to provide more logging and information about response from each requests. For setup guide, see: https://github.com/adrianmo/jmeter-backend-azure

Once added to the JMeter confugration file, you can query logs of new Azure Load testing run in real time from Application insights

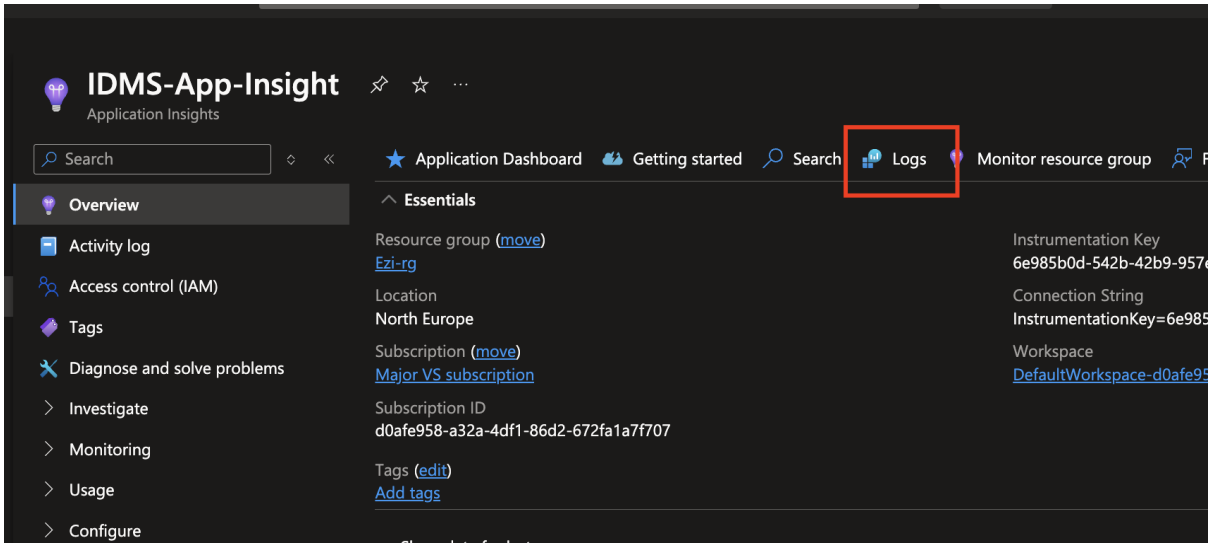


Figure 22: Application Insights logs

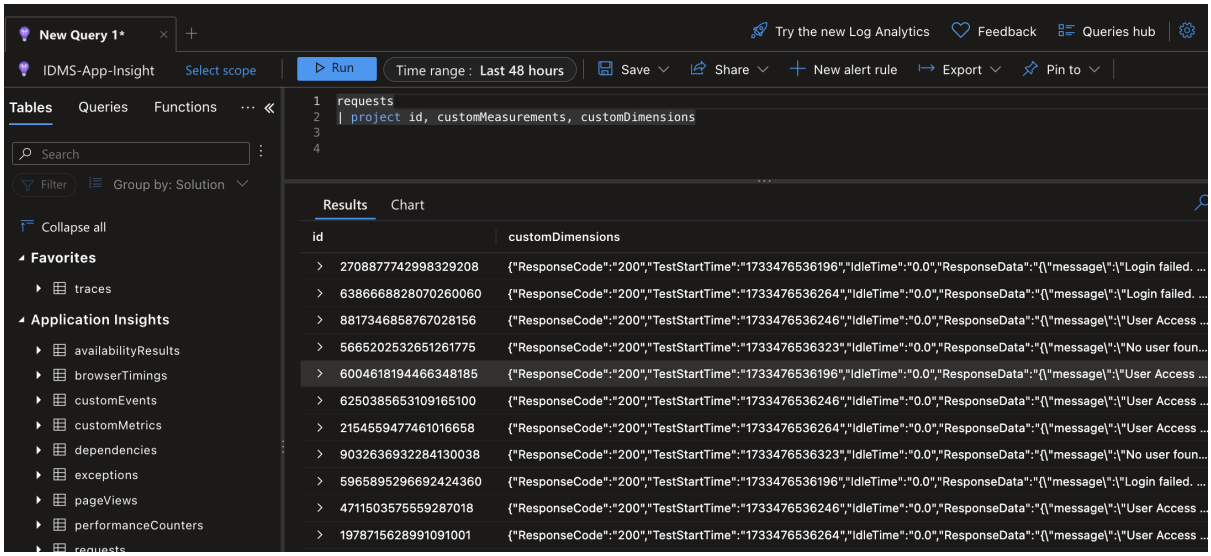


Figure 23: App Insights Logs