Use Cases and Logical Architecture

* X00130180
* Jordan Williams
* Leveraging Istio to Enforce Rules-based Authorization

## Section 1: For Each Use Case:

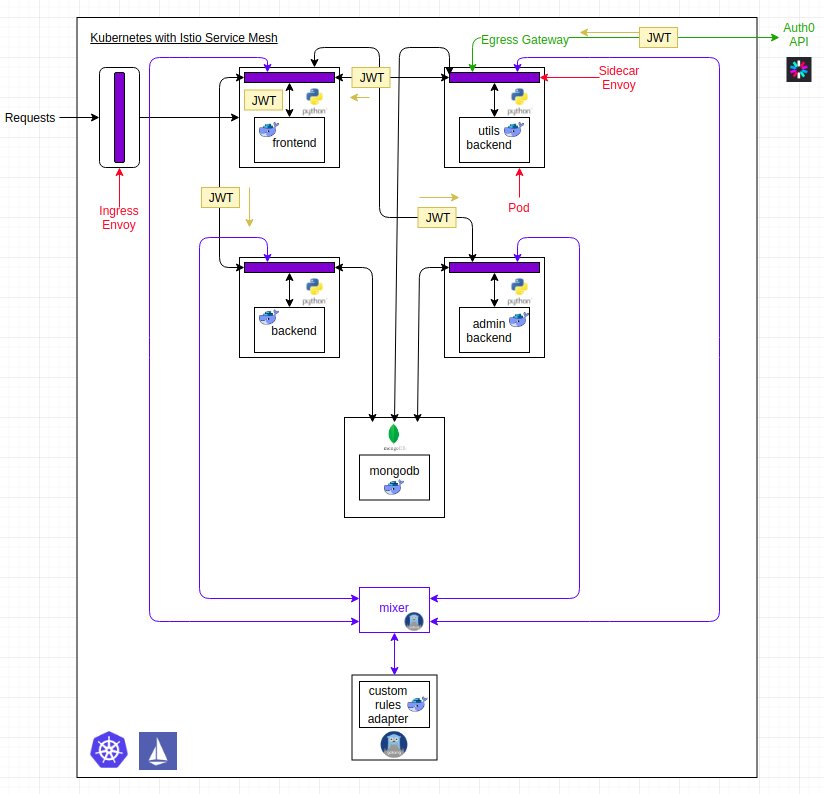
|  |  |
| --- | --- |
| Title (goal) | View Homepage |
| Primary Actor | User |
| Story | Any registered user will be able to view information regarding the service and navigate the UI. |

|  |  |
| --- | --- |
| Title (goal) | View Medical History |
| Primary Actor | Member |
| Story | A member will be able to view their own medical history. |

|  |  |
| --- | --- |
| Title (goal) | Update Patients Medical History |
| Primary Actor | Doctor |
| Story | A doctor with elevated privileges will be able to view and edit any of their assigned patients’ medical records. |

|  |  |
| --- | --- |
| Title (goal) | Data Restricted Access via Time |
| Primary Actor | Doctor |
| Story | A doctor can only access the data during the designated hours of 9am-5pm Monday to Friday. |

## Section 2: Deployment Diagram



## Deployment Diagram Discussion

A Watson Health application that implements rules based authorization and authentication between microservices. It uses Istio to implement policies that enforce rules upon the microservices. Such as, to access the various backend services you need to include a JWT in the request headers when you are attempting to send a request to these services, otherwise you will be denied access. There is also policies defined such as only Doctors can access some of the services. This is done by checking the scope of the JWT, you are issued this JWT when you login and your role determining which Auth0 API you will connect to.

I have also extended Istio’s current functionality and implemented some of my own rules. As Istio is written in GO, I had to write some services in go and then deploy them on

Kubernetes. An example of one rule that I wrote would be to enforce that Doctors can only access patient records during the times of 9-5.

## What is Kubernetes?

[Kubernetes (K8s)](https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/) is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

## What is Istio?

Cloud platforms provide a wealth of benefits for the organizations that use them. However, there’s no denying that adopting the cloud can put strains on DevOps teams. Developers must use microservices to architect for portability, meanwhile operators are managing extremely large hybrid and multi-cloud deployments. Istio lets you connect, secure, control, and observe services.

At a high level, Istio helps reduce the complexity of these deployments, and eases the strain on your development teams. It is a completely open source service mesh that layers transparently onto existing distributed applications. It is also a platform, including APIs that let it integrate into any logging platform, or telemetry or policy system. Istio’s diverse feature set lets you successfully, and efficiently, run a distributed microservice architecture, and provides a uniform way to secure, connect, and monitor microservices.

## What is a service mesh?

Istio addresses the challenges developers and operators face as monolithic applications transition towards a distributed microservice architecture. To see how, it helps to take a more detailed look at Istio’s service mesh.

The term service mesh is used to describe the network of microservices that make up such applications and the interactions between them. As a service mesh grows in size and complexity, it can become harder to understand and manage. Its requirements can include discovery, load balancing, failure recovery, metrics, and monitoring. A service mesh also often has more complex operational requirements, like A/B testing, canary rollouts, rate limiting, access control, and end-to-end authentication.

Istio provides behavioral insights and operational control over the service mesh as a whole, offering a complete solution to satisfy the diverse requirements of microservice applications.

## Why use Istio?

Istio makes it easy to create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, with [few](https://istio.io/docs/tasks/telemetry/distributed-tracing/overview/#understanding-what-happened) or no code changes in service code. You add Istio support to services by deploying a special sidecar proxy throughout your environment that intercepts all network communication between microservices, then configure and manage Istio using its control plane functionality, which includes:

* Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic.
* Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection.
* A pluggable policy layer and configuration API supporting access controls, rate limits and quotas.
* Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress.
* Secure service-to-service communication in a cluster with strong identity-based authentication and authorization.

Istio is designed for extensibility and meets diverse deployment needs.

Taken from [What is Istio?](https://istio.io/docs/concepts/what-is-istio/).

In particular we will be looking at the authentication and authorization features of istio, however, the other features come free when you install istio. I have them all configured, however, we will just be ignoring them as it wasn’t the scope of the project.

I have created a total of 6 microservices that all intertwine and run cohesively. These services are frontend, backend, utils-backend, admin-backend, mongo, and my custom-rules-adapter. These services have been developed with the use of Python, MongoDB, and Go. All of these services are wrapped in a docker container and deployed on Kubernetes with an istio sidecar installed, besides the mongo pod - which doesn’t have a sidecar installed as there is no need for it as we do not need to propagate the JWT token to the database.

It all begins whenever the user sends a request to the ingress gateway that then directs traffic towards my service mesh, specifically my frontend, with access to all the defined urls I placed in my ingress-gateway configuration. From there you then need to login to access the backend or admin-backend that allow you to read and write from the mongo pod respectively.

When you login my frontend sends a request to utils backend that reads from the mongo pod all of the users contained in the database and determines if the credentials are correct. There are 3 users, an Admin, Doctor, and Patient. All 3 users have different privileges as to what services they can access, this is enforced via Istio authentication and authorization. Once the credentials are validated, depending on which user you have logged in as and their role contained under that user in the database, frontend will send a request to utils-backend to generate and return a JWT token for that specific user. The role of the user will be contained in the scope of the encoded token. Istio normally does not allow any traffic outside of the service mesh unless it is told to whitelist a specific domain. So, I have whitelisted my Auth0 domain which allows any request to this domain succeed where previously it would fail. Utils-backend sends a POST request to my Auth0 domain to do a client credentials exchange with one of the three different configurations as for which API to connect to depending on the user. If successful, Auth0 will return an encoded JWT token that corresponds to my Auth0 JWK. This token once in utils-backend is then returned to my frontend where it is stored.

Once the JWT is in my frontend and we are logged in we can now access the various other backends depending on our user as each user has different access privileges. The JWT is passed in the header Authorization with each request to the various backends where it is verified by Istio that the token is indeed issued by my Auth0 domain and corresponds to the JWK I have configured Istio to cross check against. There is also role based authorization that is needed to access the admin-backend that allows you to write to the database. I have configured Istio that only JWT’s with the scope Admin or Doctor can access this service, and to add another layer of security I have extended Istio with my custom-rules-adapter to enforce my own custom rule that those with the scope of Doctor can only access the backend and admin-backend between the times of 9-5.