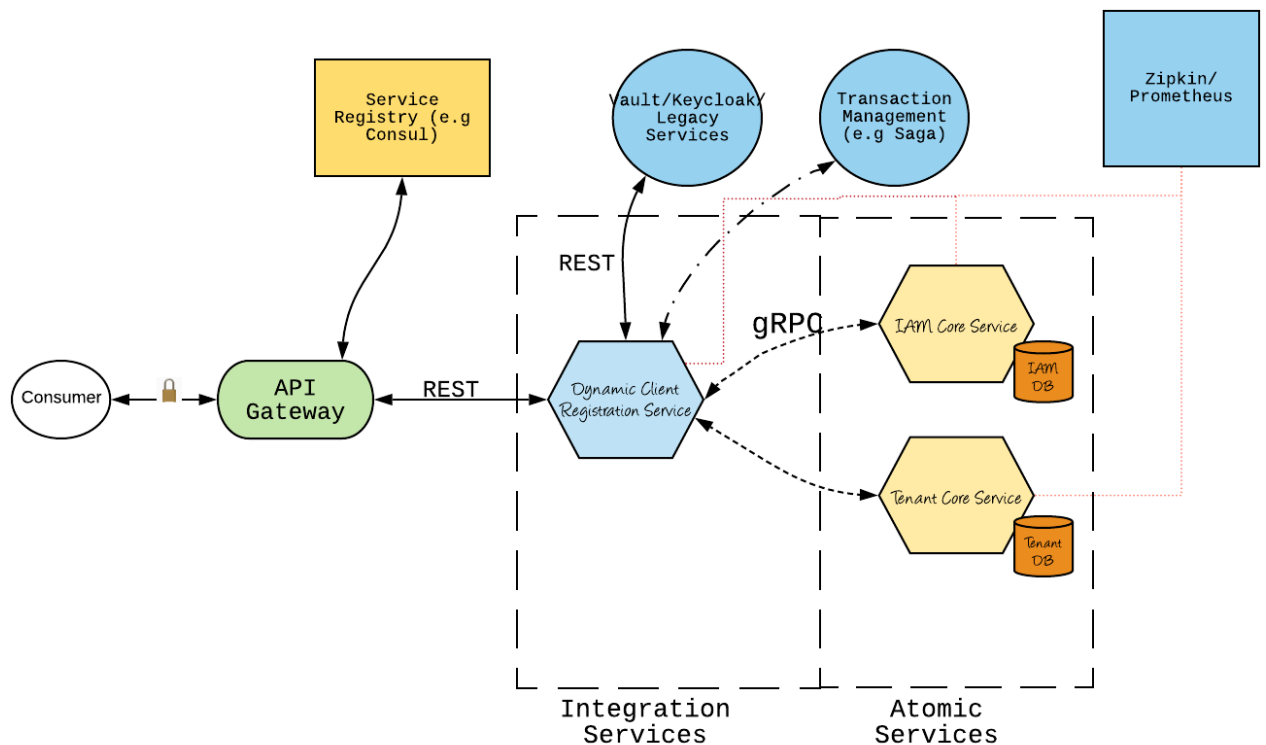


Custos Implementation Details



Types of Services

- Atomic

Atomic services are independent services with minimal network communication and they have their own database with local data. These services focus on tiny specific tasks bound to their context.

- Integration

Integration services represents the actual business use case and they act as orchestrators between core services and external services. Those are exposed as rest apis via API Gateway. Integration services talk to atomic services via gRPC protocol. Integration services includes gRPC stub of core services.

Deployment Archi

Each integration services and core services are deployed to kubernetes cluster as separate services and have their own service addresses. Maven build should upload docker images to custos docker repository. Custos helm chart fetches relevant docker images from docker repository and deploy into kubernetes cluster.

Transaction Management

Main problem with this architecture is transaction management and it is planned to handle with Saga using Axon Server or Camunda like framework. All the services are written using Spring Boot and Spring Cloud.

API Management and Service Discovery

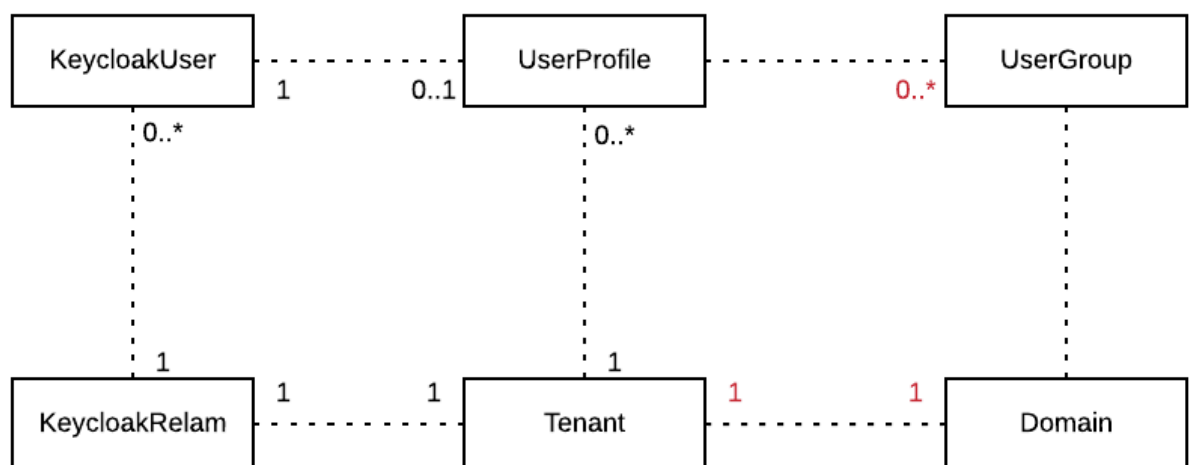
All the Integration services are registered with the service registry such as Consul. API Gateway is used for API Management capabilities such as SSL Termination, throttling..etc. There are Open Source API Gateways such as Ambassador, Envoy which are compatible with kubernetes and service registries. API Gateway is responsible for service discovery.

Observability

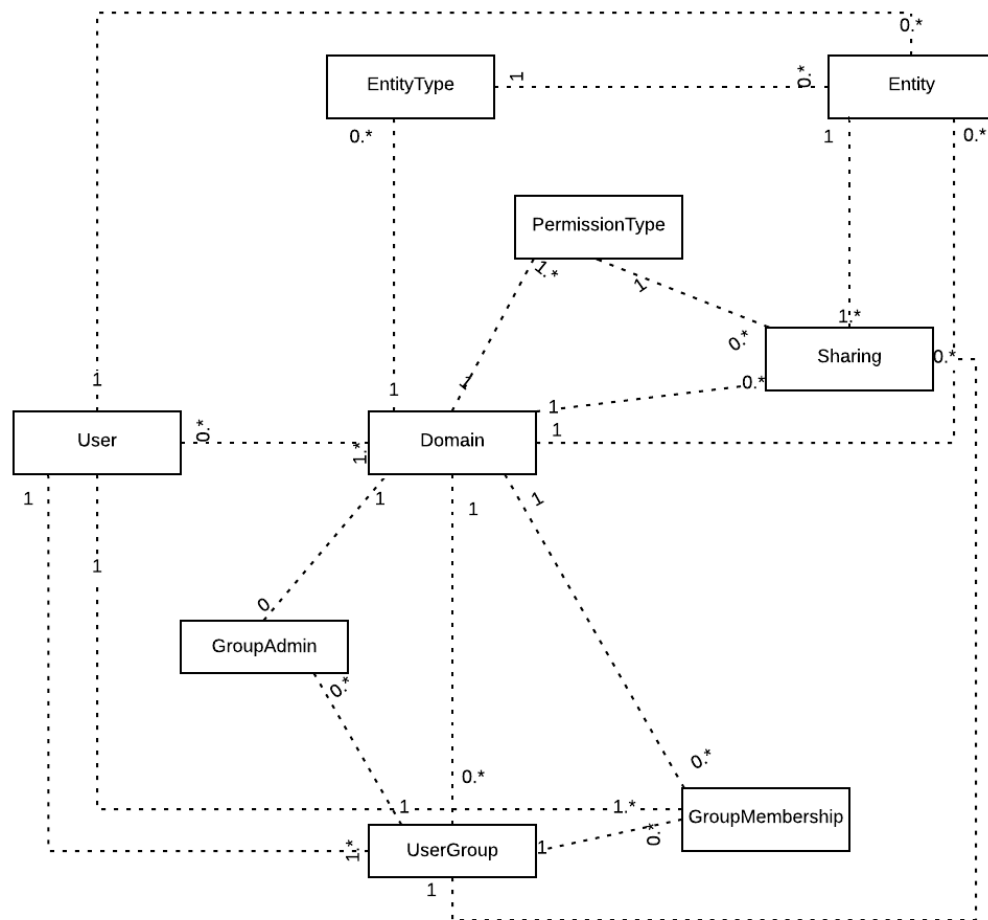
Mainly we need to consider three pillars, logging, metrics, and tracing. For logging purposes, SLF4J can be used and obtain local logs. For tracing the message from end to end we can integrate Spring-Cloud-Sleuth and Zipkin servers. Sleuth acts as a client on every node and sends trace information to Zipkin server. For collection of metrics Prometheus can be used.

DB Layer Design

- Custos Tenant Profile/ User Profile Services



- Custos Group Management and Sharing Services



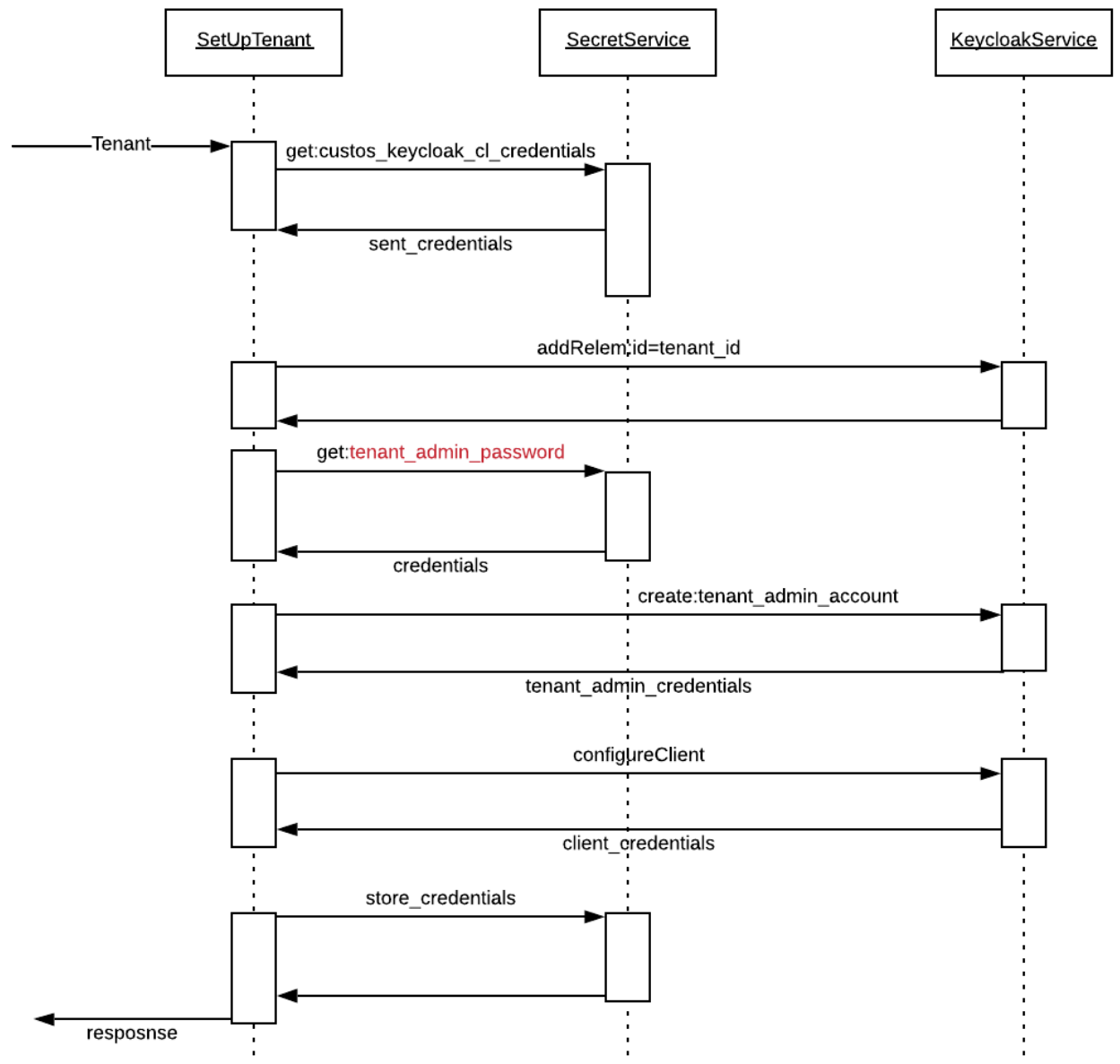
Group Types = {USER, GROUP}

Group Cardinality = {SINGLE, MULTI}

Sharing Types = {Direct Cascading, Indirect Cascading, Direct Non Cascading}

- Setup Tenant Client

IAM Admin Service: Tenant Registration Flow



Data Persistence Implementation

API Version Management in Rest

Versioning through URI Path

<http://custos.scigap.org/v1.0.0/tenant>

- Major.Minor.Patch version
Route to the endpoint based on version

Versioning through query parameter

<http://custos.scigap.org/tenant?version=1.0.0>

Versioning through Header

Accepts-Version:1.0.0

Versioning through content negotiation

“Accept: application/vnd.xm.device+json; version=1”

High cost of development with more granular versioning in resource level

Advantages

- Easily generate client stubs using swagger or similar tools
- Support highlevel hypermedia
- Human readable and easy to debug
- standardized

Drawbacks

- HTTP Caching may have problems

gRPC as Public Endpoints

-

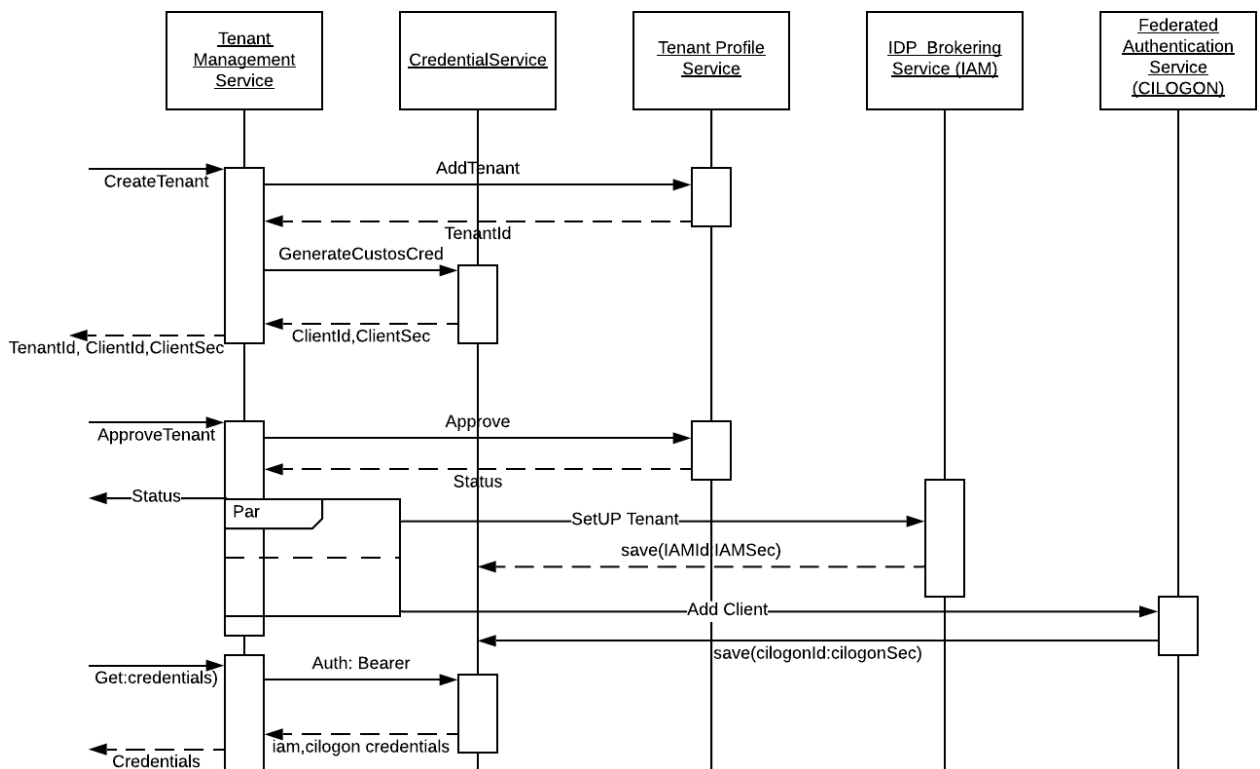
Instrumentation With Prometheus

Endpoint :- `actuator/prometheus`

Vault and Keycloak are deployed in Kubernetes with HA[1].

Tenant Management Service

- Create Tenant Loop



Graceful Deployments and Updates [3,4, 5]

- Recreate
Complete destroy existing pods and recreate new pods. Does not provide zero down time

- Rolling Update

Allow kubernetes to handle rolling pods with configurations. This may require backward/forward compatibility of APIs.

- Canary
Create route rule both versions and scaling is done by weighting.
- A/B Testing
Route traffic based on load balancer headers. It requires smart load balancer
- Blue green deployment

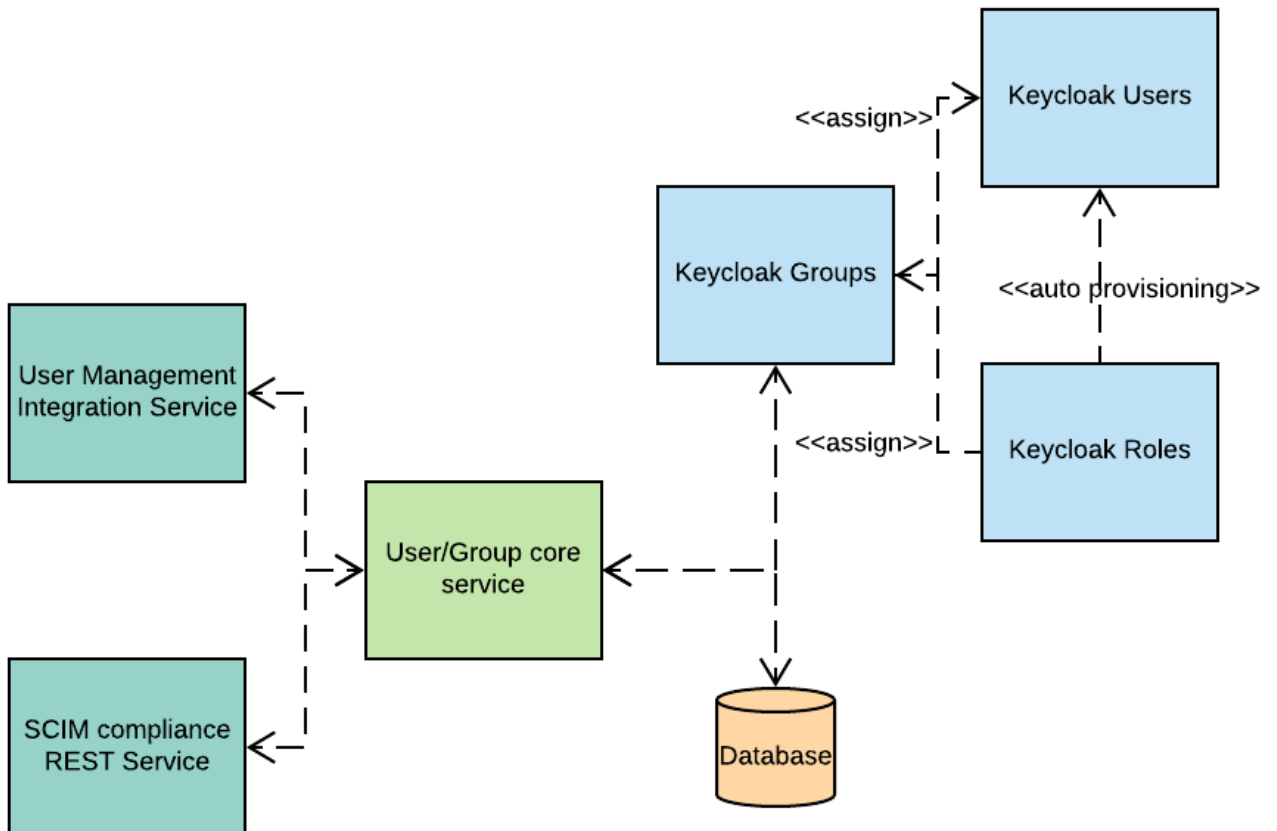
Another solution could be that you run two production environments. You always deploy to only one of them, and you only point your load balancer to the new one after you have verified that the new version works as it is expected.

- Shadow

Deployment mirroring. Setup is expensive and requires a smart load balancer like Isitio.

Groups Implementation

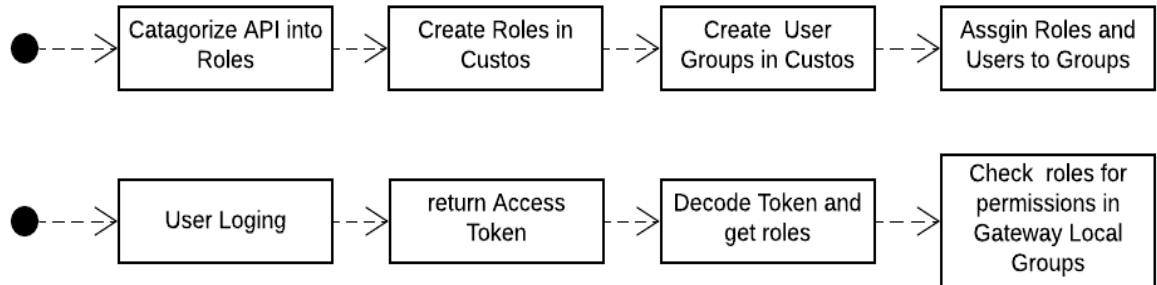
- Architecture



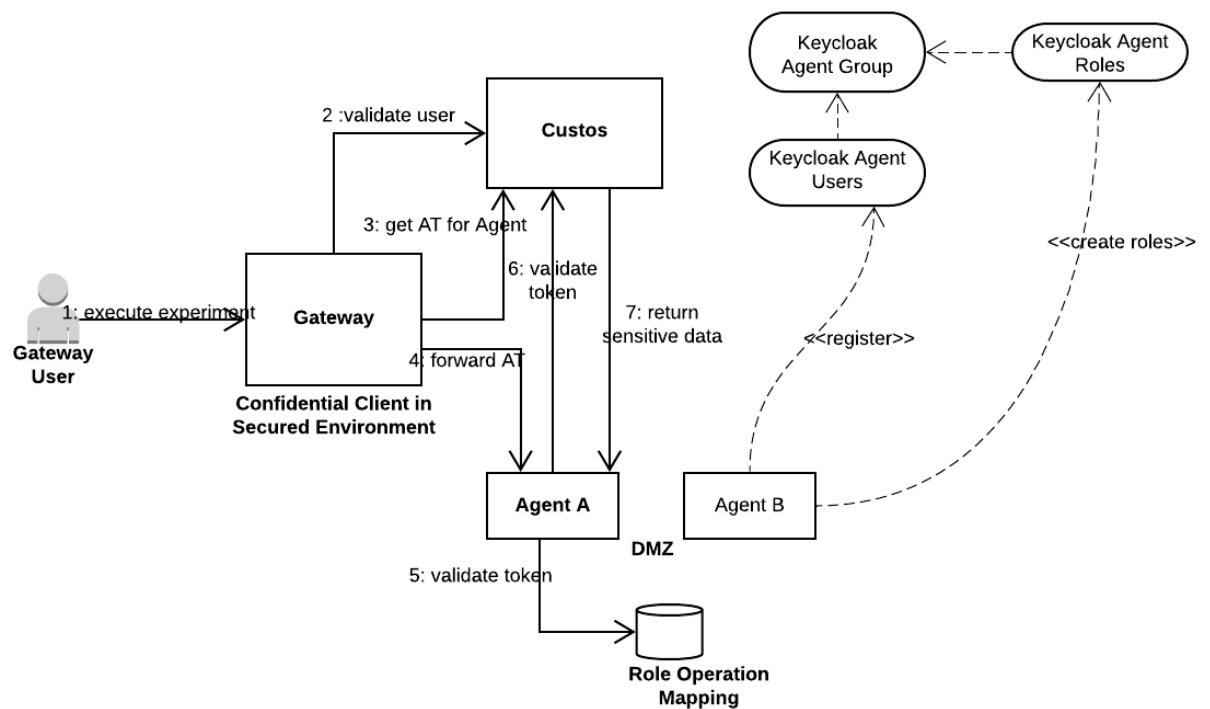
- User Management Integration service is used to access user and group related operations. This can be accessed via REST clients, python SDK, or Java SDK.
- SCIM compliance REST service expose REST endpoints compliance to [\[1\]](#)
- User/Group core service manages internal Custos database and maintains relevant data and metadata.
- Keycloak Groups are created for each Group created in Custos. Keycloak Roles can be assigned to Keycloak Groups and all the members in that Group are automatically assigned those roles. Moreover, hierarchical grouping can be used with Keycloak Groups. Attributes and Roles assigned to parent groups are automatically inherited to Child Groups.

- Use cases

- API Authorization



- Distributed Agent Authentication and Authorization



- Any gateway having distributed agents that are operated in non secured environments which need to access sensitive data related to Gateway should be handled in a secured manner.
(e.g MFT agents in Airavata)
- Gateway can register its Agents as users in Custos and those users are added to a particular Group.
- These agents may have different operations, they are categorized based on Roles and created in keycloak as in the above example.

- Gateway creates a separate client to obtain short lived access tokens for Agents

Execution Steps

1. User login to Gateway and Gateway has the User AT.
2. User executes an experiment, subsequently Gateway authorizes the user with provided AT .
3. To execute the experiment, Gateway selects a Distributed Agent to copy input data to output location.
4. The Gateway fetches short lived AT for agent and forwards the AT to agent with operation metadata.
5. Agent decodes the token and checks whether the token has relevant roles embedded to proceed with the operation.
6. The Agent needs sensitive data (E.g SSH Credentials) for execution.
7. Agent requests sensitive data from Custos by providing AT and specific claims to validate itself to Custos.
8. Custos validates Agent and releases data.

- SCIM Implementation [\[1\]](#)

- Media Type : application/scim+json
- Schema endpoint should be exposed. Should prefix by "urn:ietf:params:scim:api:"
- Endpoints
 - /Users
 - /Groups
 - /ServiceProviderConfig
 - /Me
 - /ResourceTypes
 - /Schemas
 - /Bulk
 - /.search
- Attribute Types :- (read only, read write)
- Response should contain Location header for create requests and HTTP Status shall be 201
- Pagination Request with startIndex, count, Response with "totalResults", "itemsPerPage", "startIndex"

Existing SCIM Implementations

- WSO2 Charon
- [SCIM2 SDK](#)

- [2]<https://medium.com/@sankar.p/how-grpc-convinced-me-to-chose-it-over-rest-30408bf42794>
- [3]<https://kubernetes.io/blog/2018/04/30/zero-downtime-deployment-kubernetes-jenkins/>
- [4]<https://medium.com/platformer-blog/enable-rolling-updates-in-kubernetes-with-zero-downtime-31d7ec388c81>
- [5]<https://www.cncf.io/wp-content/uploads/2018/03/CNCF-Presentation-Template-K8s-Deployment.pdf>