

Building stateless SSO using
Keycloak in multi tenant model

Speaker Info

.Mridul Deka -- Technical Architect

.Parshotam Singh – SME

.Johnson Controls(India) -- <https://www.johnsoncontrols.com/>

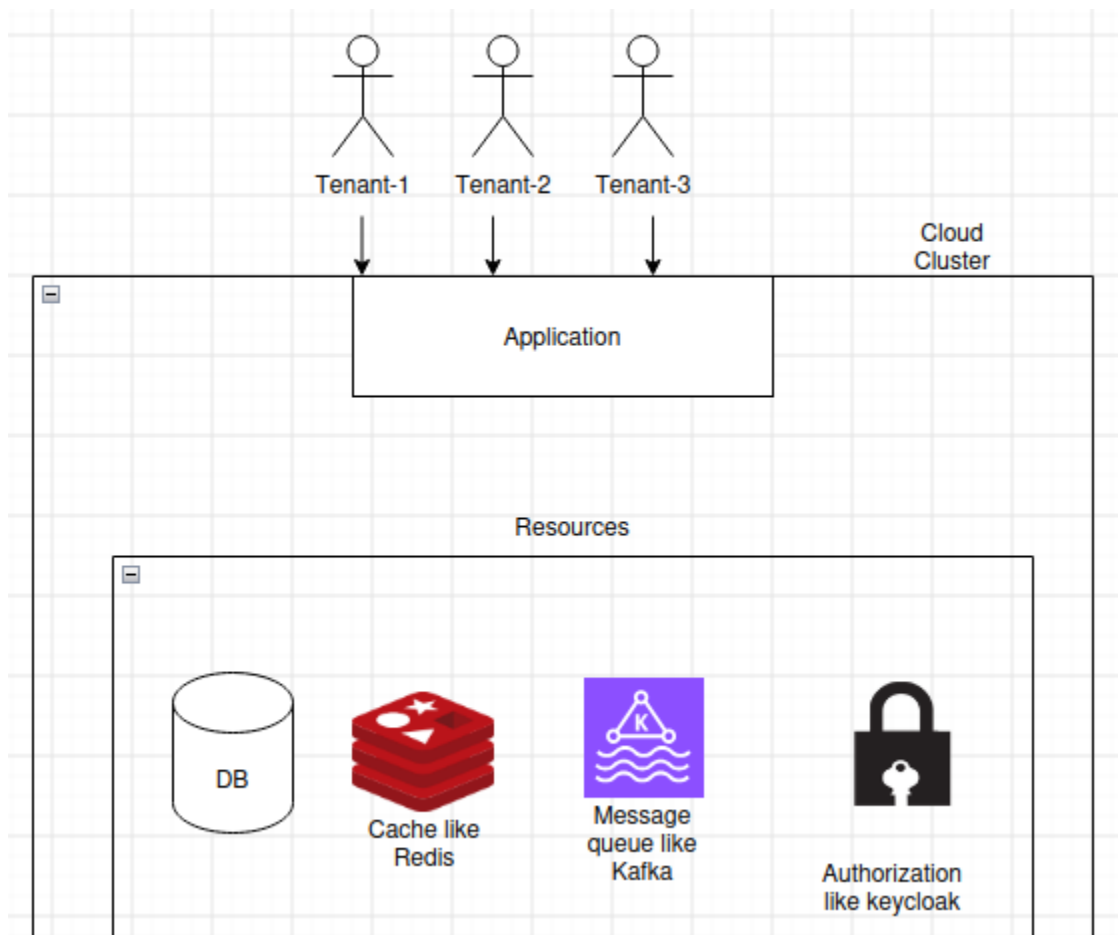
Keywords

- Stateless SSO
- Multi Tenant Model

Context

- What is multi tenant architecture and why is it required ?
- What is tenant , is it customer ?
- Lets say there is a small start up which is using keycloak during early stage of product building. It decides to go with shared tenant model.

Shared Tenant model



Shared Tenant Model

- Multiple Tenants or customers sharing same infra.
- One instance of Keycloak running in a single K8S cluster.
- For every tenant, separate realm set up in single keycloak instance.
- N tenant : N realm mapping in keycloak.

Pros and Cons of Shared tenant model

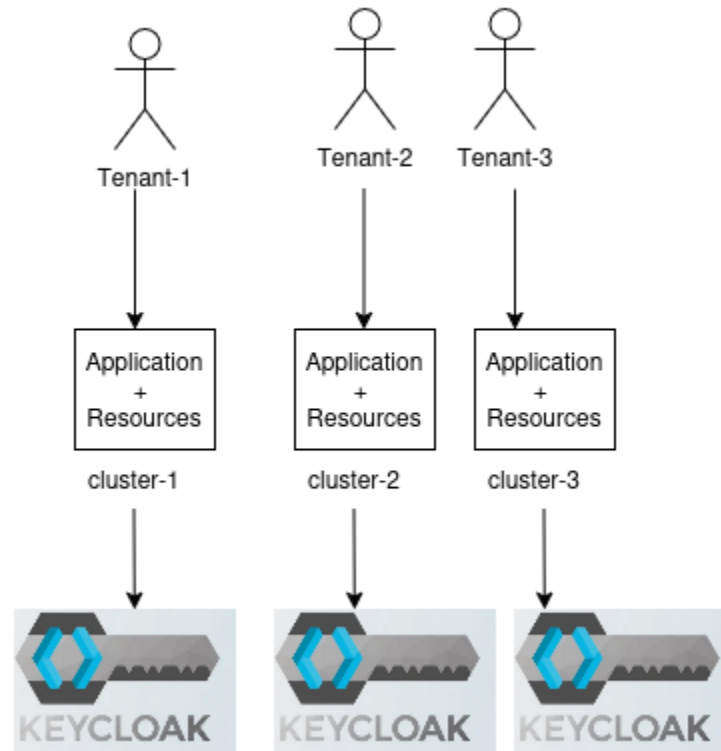
.Pros --

- Design is simple as all tenants are set up in single cluster.
- Works well for small start-ups who want to onboard customers quickly.

.Cons --

- Difficult to manage as number of customers increases.
- Not a scalable solution.
- Becomes single point of failure. If single keycloak cluster is down, all customers gets impacted. There will be a global outage.
- What if there are customer specific features.

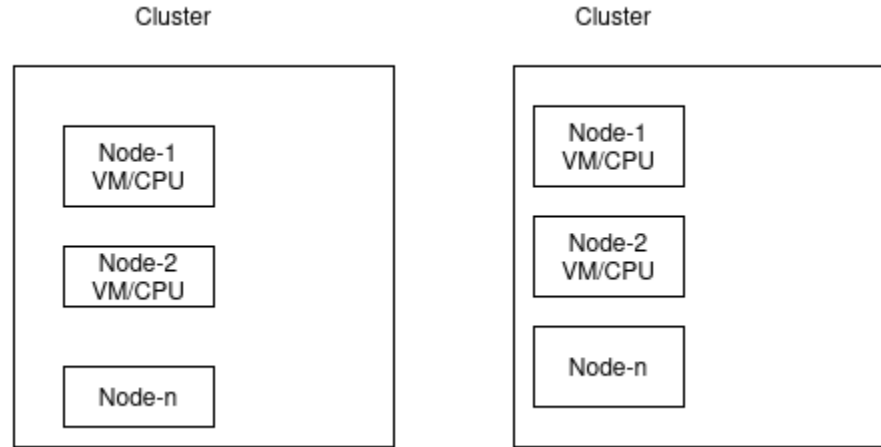
Single Tenant Model



Single Tenant Model

- Each tenant has its own keycloak instance. Each tenant is deployed in a K8S cluster.
- For each tenant, separate realm is created.
- 1 tenant : 1 realm mapping in keycloak.
- Pros --
 - Isolation of tenants and its infra like keycloak, more modularity.
 - Outage is at local level not global level.
- Cons --
 - Cloud cost** can increase significantly if there is dedicated cluster for each tenant.
 - Difficult to manage as number of customers increases. There can be n-tenants and n can be a very large number.

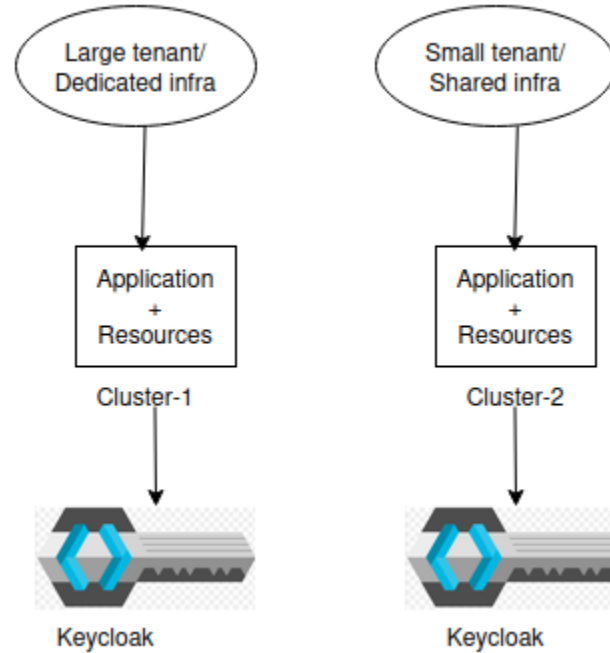
Understanding Cloud Cost



General Thoughts on cloud cost

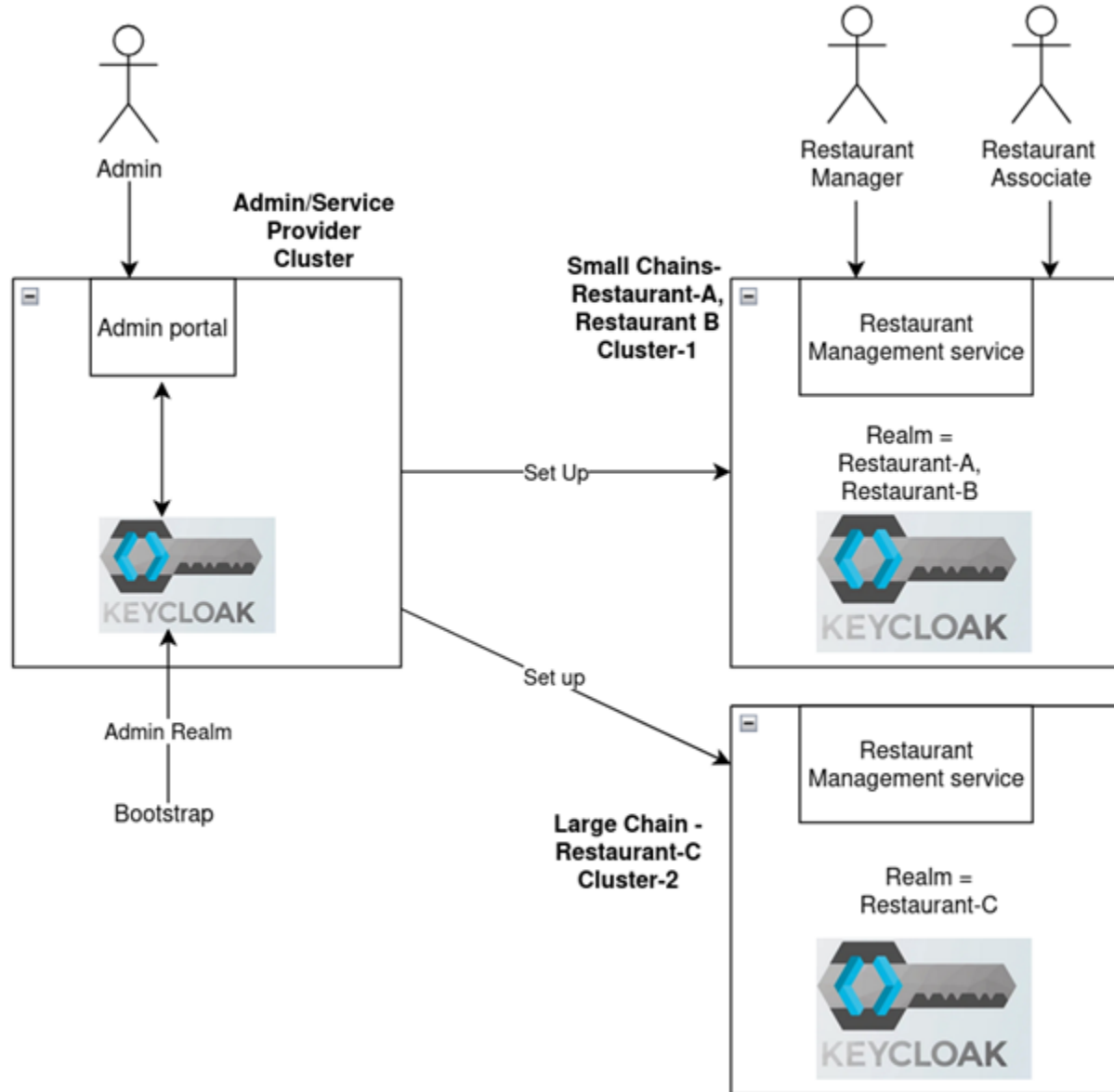
- If nodes keeps increasing in a cluster, cost will increase. If nodes remain constant but cluster keeps increasing, cost will increase.
- Do we need **separate keycloak instance** for each tenant?
- Does single tenant model approach differs for B2B and B2C apps ?
- How are users on boarded in keycloak in single tenant architecture ?

Proposed Solution for previous model



Use Case for multi tenant/realm strategy

- Lets consider your org provides a platform for a food delivery app where all restaurants in a particular location can be displayed to users.
- Users can select restaurant and place order for food from there account. (B2C)
- Restaurant workers can manage orders from their account using Restaurant Management System. (B2B)
- We will consider **Restaurant management system** application where restaurant manager and associates can login and manage orders from users.



Pros and Cons of Hybrid Architecture

-Pros --

- Small group of customers/tenants having low data volume(like small number of users) can be deployed in a same cluster sharing same Keycloak instance. For ex – 3 new restaurant open in a region having just one center with small capacity to operate.
- Minimize risk of global outage, more chances of cluster specific(**local**) outage.

-Cons --

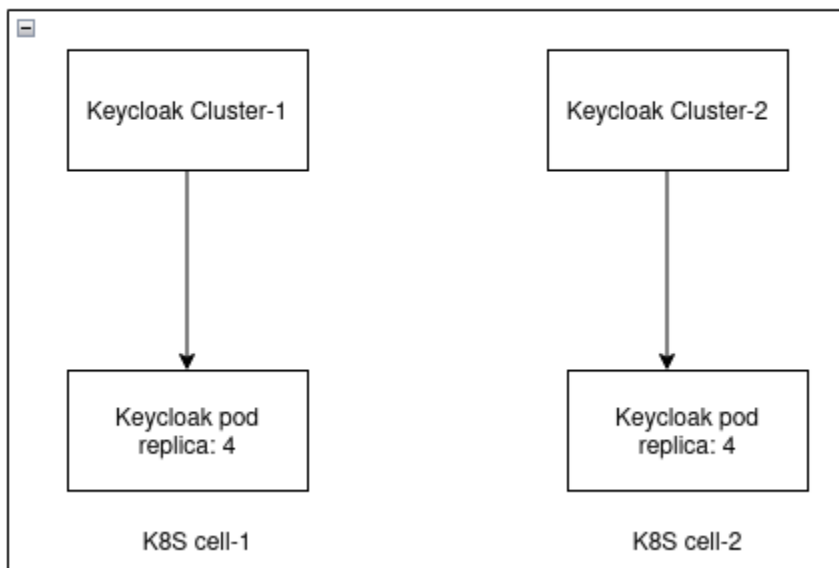
- Dedicated cluster** still required for large customers(having large no of users) but more cluster means more customers which increases business. Can we optimize this solution further ?

Cell Based Architecture

- Instead of cluster, partition can be done in a cell.
- Each cell is an isolated workload instance containing dedicated resources like keycloak, DB, microservices etc.
- Tenants or Realms can be deployed at cell level. Large customer can have dedicated cell and small group of customers can be deployed in a single cell.
- Cell boundaries can be logical like individual K8S name-space, K8S cluster or at hardware level like node groups or at infrastructure level like Availability Zones(AZ), data center regions.

Cell Based Design

Managed K8S Engine



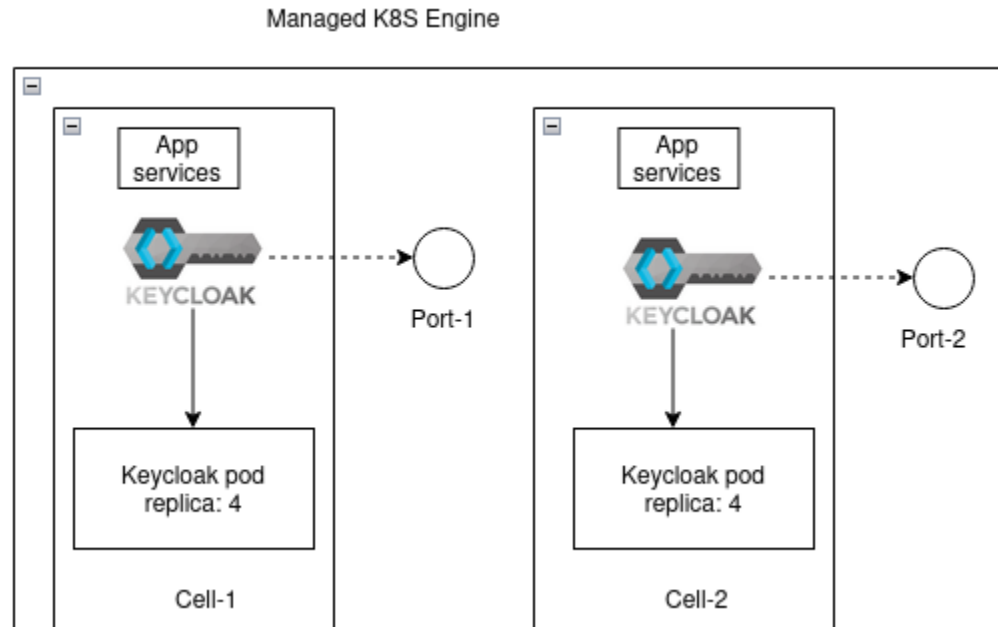
Cell Based Workflow

- .Create a cluster inside your managed K8S engine.
- .In this example, we take K8S namespace as cell. We will use minikube to implement local K8S cluster and cells under it to provision keycloak instance.
- .Create a cell(namespace) <dedicated-cell> for a large tenant--(tenant-1) which will use dedicated keycloak instance.
- .Create a cell <shared-cell> for group of small tenants-- (tenant-2,tenant-3,tenant-4) which will use dedicated keycloak instance.
- .Keycloak pods inside cell are running as replica set.

Minikube Demonstration

Namespaces		
Name	Labels	Phase
 ingress-nginx	app.kubernetes.io/instance: ingress-nginx	Active
	app.kubernetes.io/name: ingress-nginx	
	kubernetes.io/metadata.name: ingress-nginx	
 kubernetes-dashboard	addonmanager.kubernetes.io/mode: Reconcile	Active
	kubernetes.io/metadata.name: kubernetes-dashboard	
	kubernetes.io/minikube-addons: dashboard	
 dedicated-cell	kubernetes.io/metadata.name: dedicated-cell	Active
 shared-cell	kubernetes.io/metadata.name: shared-cell	Active

Cell Based Design in same cluster



Keycloak in dedicated-cell

Manage realms

 Search



Create realm



 Refresh



Name

Display name



master

Current realm

Keycloak



Tenant-X_large

—

Keycloak in shared-cell

Manage realms

Create realm

☐	Name	Display name
☐	master Current realm	Keycloak
☐	Tenant-C_small	–
☐	Tenant-A_small	–
☐	Tenant-B_small	–

Stateless SSO Motive

.Stateful SSO : User state like username, email, password etc. is stored in system.

.Problem with Stateful SSO :

.Handling user information and passwords within infrastructure presents issues related to PII (Personally Identifiable Information) and GDPR adherence.

.Passwords must adhere to policies and guidelines such as a minimum of 8 characters, regular password changes, and not matching the last three passwords, etc.

.When a user departs from the organization, the system must remove user records from our infrastructure.

.SSO service should focus purely on the user authentication rather than user storage , we created a system that is scalable, easier to maintain, or more secure.

Using Keycloak as Identity broker for Stateless SSO

