**Deployment of Mattermost with Multi-Community Subdomain Management on Kubernetes**

**1.Introduction**

**Objective:**

This document provides step by step guide to deploying mattermost on Kubernetes cluster while also managing multiple sub domains dynamically using admin controller.

**2.System Architecture**

**Architecture overview:**

The deployment requires following components that are mandatory to make this project works:

**1.Kubernetes cluster(EKS,GKS):** where mattermost platform hosted.

**2.NGINX Ingress controller:** which is helpful in routing traffic based on hostnames.

* It also supports host based, path based and regex based routing
* While using nginx is cost efficiency because we have to just pay for pods running
* It automatically create sub domains
* Full control over ingress rule
* It also lower cost hence it runs inside cluster, compare to multiple load balancer provided by cloud.
* Assing one external load balancer for distribute traffic.

**3.Certificate manager**: This automates SSL certifications manually.

* It uses FASTAPI which handles API request dynamically,also support data validation using Pydantic model.

**4.Admin controller**: Dynamically create ingress rules for new subdomains.

* Automatically create DNS records for new communities.
* Generate Ingress rules dynamically for each subdomain.
* Issue SSL certificates for new subdomains using Cert-Manager.

**3. Prerequisites and Setup**

**a. Prerequisites:**

1. Kubernetes cluster with minimum of 3 nodes.
2. Helm installed, because it acts as package manager and has pre configured templates which makes works easier to write YAML files.
3. Domain name with wildcard DNS configured.
4. Why wildcard DNS:

* Wildcard DNS allows all subdomains of a domain to resolve to the same IP address without needing to create individual DNS records.
* Example: 192.168.1.100 -> community1.com
* Example: 192.168.1.100 -> community2.com
* Even we can setup wildcard DNS using cloudflare, AWS Route53

1. Encrypt certificate provider.

**b. installation steps:**

**Step 1: Setup Kubernetes cluster**

* kind create cluster --name mattermost-cluster

**Step 2: Install NGINX Ingress controller**

* helm install ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --create-namespace

**Step 3: Install cert manager for SSL management**

* kubectl apply -f https://github.com/jetstack/cert-manager/releases/latest/download/cert-manager.yaml

**Step 4: Deploy Mattermost using Helm**

* helm repo add mattermost https://helm.mattermost.com
* helm install mattermost mattermost/mattermost-team-edition.

**4.Admin Controller implementation**

**How admin controller works**:

1. Receives an API request to add a new community subdomain.
2. Generates an Ingress YAML definition dynamically.
3. Applies the generated Ingress resource to Kubernetes.
4. Enables SSL certificates automatically using Cert-Manager.
5. Deploy admin controller in Kubernetes

* kubectl apply -f admin-controller.yaml
* admin controller will watch for CRD objects
* Generate NGINX ingress rules dynamically
* Update dns records
* Request SSL certificate using Cert manager
* Delete sub domain if it community is removed.

**Example of how it generate ingress yaml definition dynamically:**

def create\_community(community\_name: str):

ingress\_yaml = generate\_ingress\_yaml(community\_name)

with open(f"{community\_name}.yaml", "w") as file:

file.write(ingress\_yaml).

**5.Traffic routing and security**

**1. Ingress controller configuration:**

* NGINX controller will route based on subdomains.

Example for creating and configuring ingress controller:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: mattermost-ingress

spec:

rules:

- host: community1.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: mattermost-service

port:

number: 8065

In above code I have just written as example the code is not fully complete, as I have shown for demo purpose what will b the YAML structure for declaring nginx controller.

**6.Scaling subdomains**

* Dynamic Subdomain Creation: The Admin Controller automates Ingress rule generation.
* Load Balancing: The NGINX Ingress Controller distributes traffic efficiently.
* Database Scaling: PostgreSQL/MySQL is used with replication.

**7.Testing and Validation**

* curl -X POST http://admin-controller-service/add-community/community1
* kubectl describe certificate community1-tls
* curl -k https://community1.example.com

**Note:** We have to create issuer. yaml file which helps requesting SSL/TLs certificates from CA, without cert-manager it will not generate certificates automatically.