MICROSERVICES ARCHITECTURE DESIGN AND IMPLEMENTATION

Muhammad Naeem, Akhtar | Cloud Engineer

REPLY

MEETING AGENDA

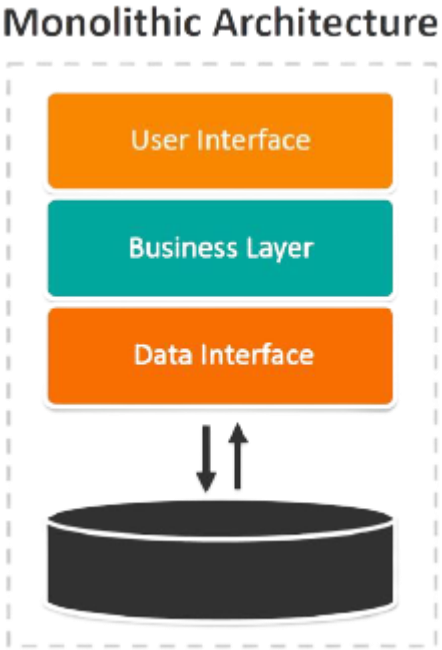
MICROSERVICES ARCHITECTURE DESIGN AND IMPLEMENTATION

1. Traditional way of applications architecture
2. Microservices architecture
3. Kubernetes
4. Demo
5. MS on AWS
6. MS on OCI
7. MS on azure

TRADITIONAL WAY OF APPLICATIONS ARCHITECTURE

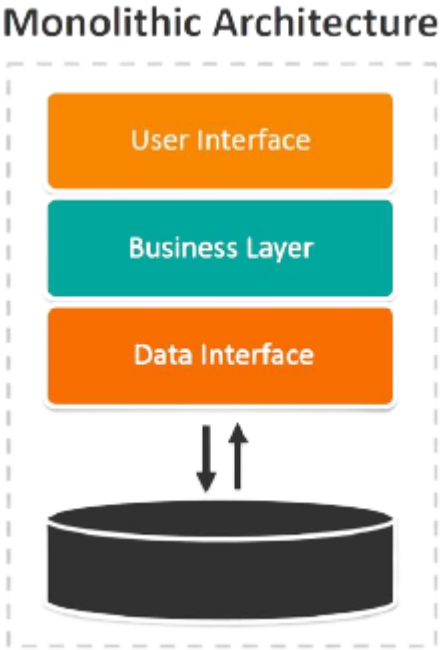
MONOLITHIC ARCHITECTURE

- Traditional way/architecture of computing is also known as Monolithic computing
- As a definition word monolithic means **very large, united, and difficult to change**
- A monolithic architecture is the traditional unified model for the design of a software program
- Monolithic, in this context, means composed of all in one piece



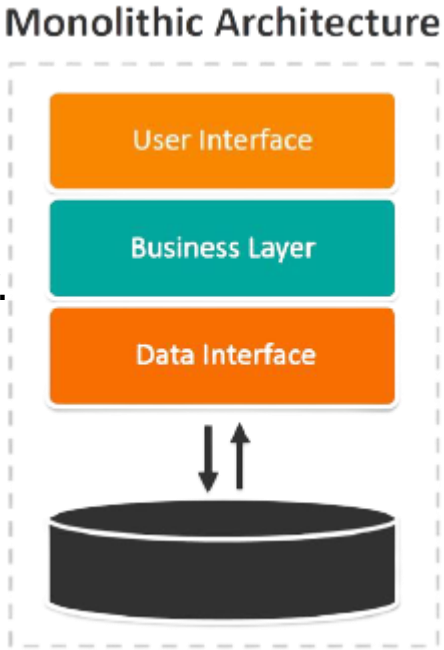
MONOLITHIC ARCHITECTURE

- Components/Layers of the program are interconnected and interdependent
- Monolithic software is designed to be self-contained meaning having all that is needed, in itself
- In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled
- Application is too large and complex to fully understand and made changes fast and correctly



MONOLITHIC ARCHITECTURE

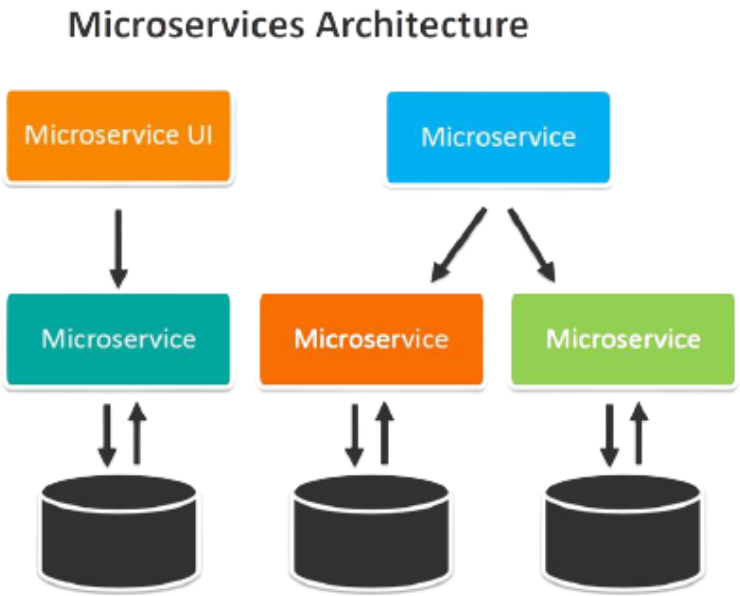
- You must redeploy the entire application on each update
- The size of the application can slow down the start-up time
- Another problem with monolithic applications is reliability. Bug in any module can potentially bring down the entire process
- Monolithic applications has a barrier to adopting new technologies. Since changes in frameworks or languages will affect an entire application it is extremely expensive in both time and cost



MICROSERVICE ARCHITECTURE

MICROSERVICE ARCHITECTURE

- The microservices architectural style is an approach to developing a single application as a suite of small services
- Each runs in its own process and communicates with lightweight mechanism, often an HTTP resource API

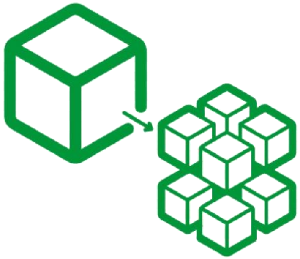


MICROSERVICE ARCHITECTURE

ADVANTAGES

Microservices do have distinct advantages:

Better Organization



Decoupled

Better Performance

MICROSERVICE ARCHITECTURE

ADVANTAGES: BETTER ORGANIZATION

Better Organization:

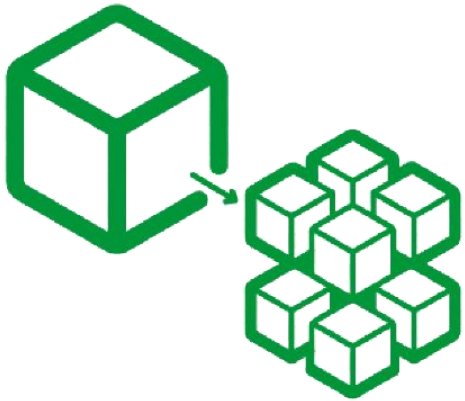
- Microservice architectures are typically better organized
- Each microservcie has a very specific job, and it is not concerned with the jobs of other components

MICROSERVICE ARCHITECTURE

ADVANTAGES: DECOUPLED

Decoupled :

- Decoupled services are also easier to change, update and re-configure to serve the purpose of differen apps
- They also allow for fast, independent delivery of individual parts within a larger, integrated system



MICROSERVICE ARCHITECTURE

ADVANTAGES: BETTER PERFORMANCE

Better Performance:

- Under the right circumstances, microservices can also have performance advantages depending on how they're organized
- It's possible to isolate hot services and scale them independently of the rest of the app

MICROSERVICES COMPLEXITIES

- In bigger microservices architecture it is very difficult to configure, manage, and keep the whole system running smoothly
- Adjusting microservices in a manner where you can achieve max hardware resources utilization is also a challenge
- Making sure all services are working is also no easy
- Above all scaling those

MICROSERVICES COMPLEXITIES

- Doing all this manually is hard work
- We need automation, which includes
  - Automatic scheduling of those components to our servers
  - Automatic configuration
  - Supervision
  - Failure-handling
- This is where **Kubernetes** comes in

KUBERNETES

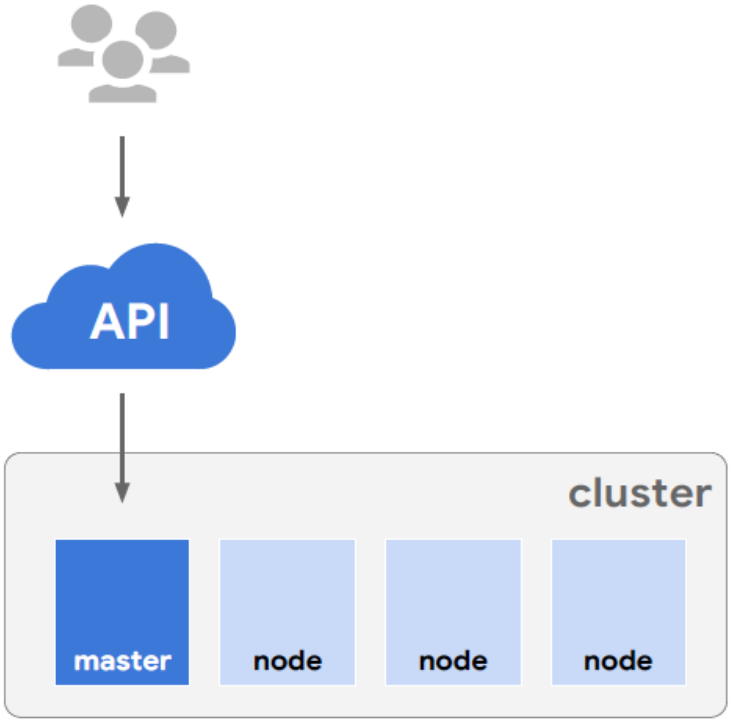
KUBERNETES

- Kubernetes enables developers to deploy their applications themselves and as often as they want, without requiring any assistance from the operations (ops) team
- Kubernetes doesn’t benefit only developers. It also helps the ops team by automatically monitoring and rescheduling those apps in the event of HW failure
- Kubernetes abstracts aways hardware infrastructure and exposes your whole data center as a single enormous computational resource

KUBERNETES

- When you have multiple servers and you are deploying a multi-component application through Kubernetes
- This easy component management makes Kubernetes great for most on-premises data centers as as for cloud providers
- Kubernetes allows cloud providers to offer developers a simple platform for deploying and running any type of application, while not requiring the cloud provider’s own sysadmin to know anything about the tens of thousands of apps running on their hardware

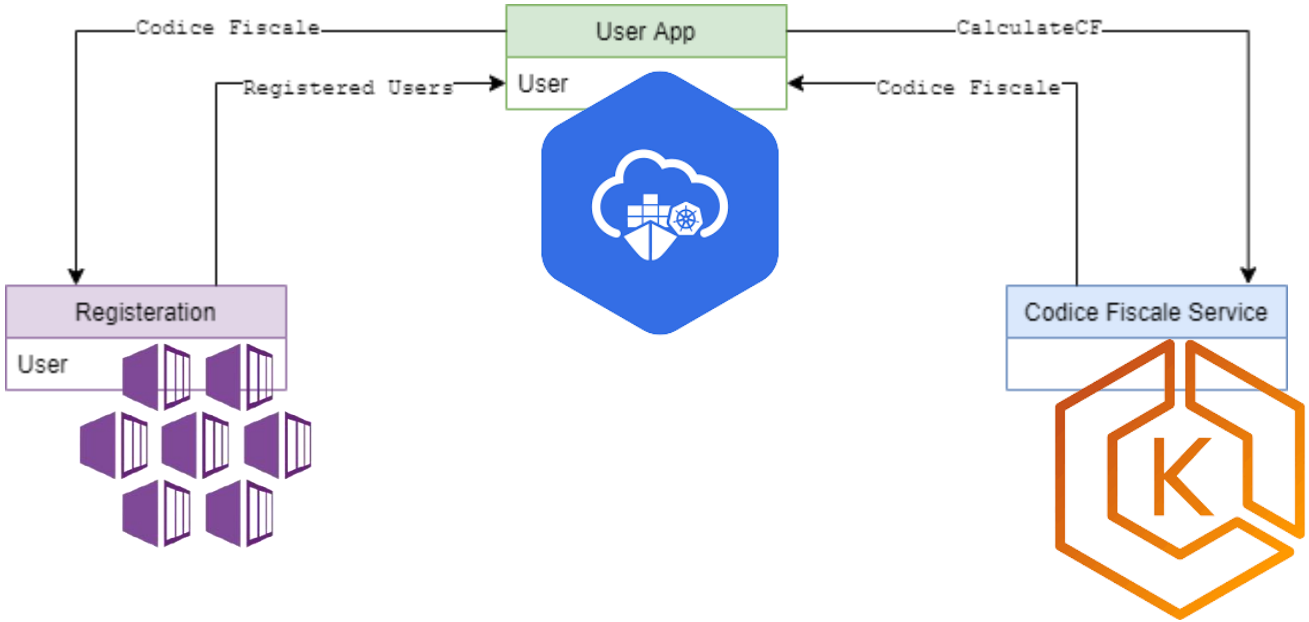
KUBERNETES



External registry

DEMO

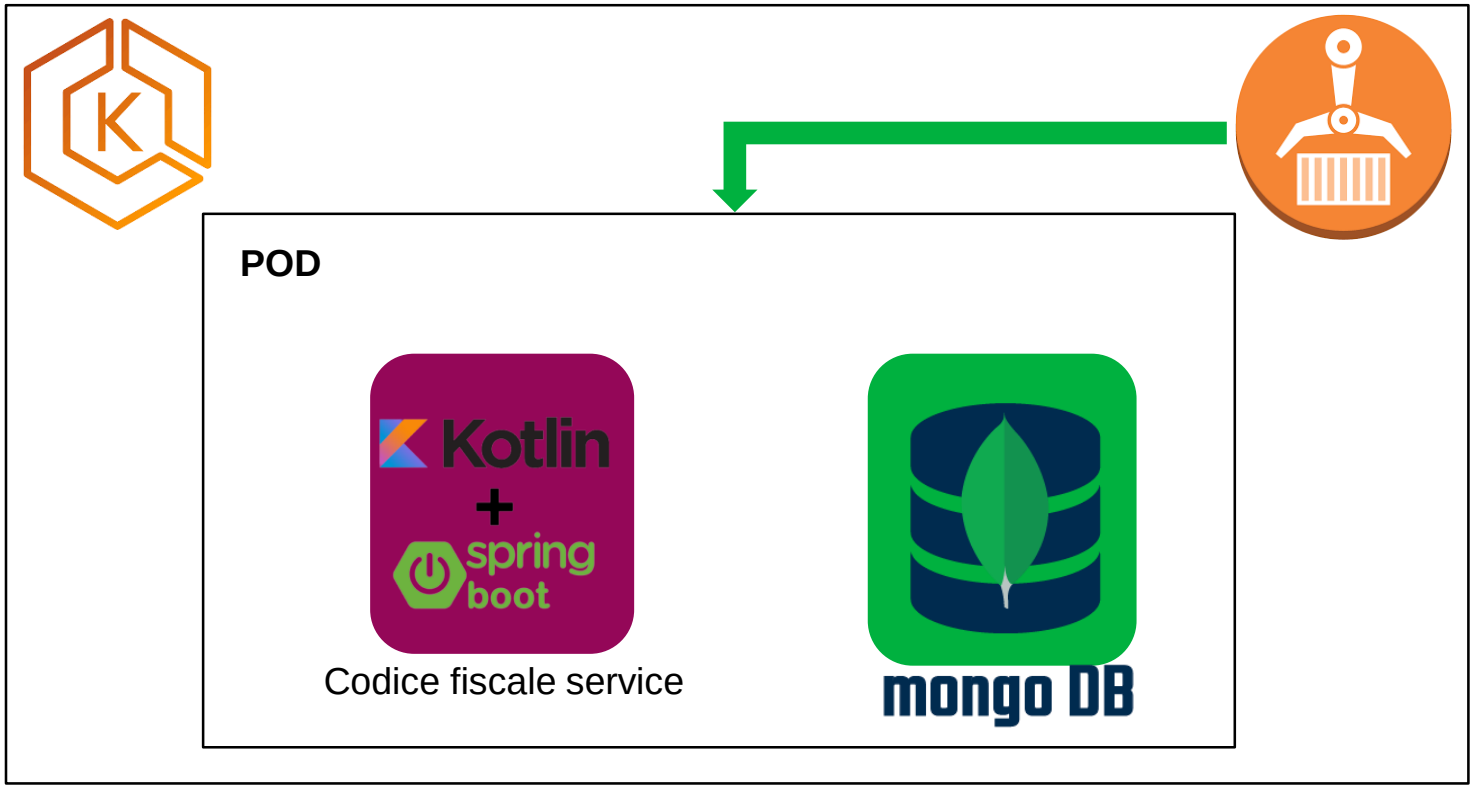
DEMO APPLICATION



CODICEFISCALE SERVICE

DEMO APP

CODICE FISCALE SERVICE



DEMO APP

CODICE FISCALE SERVICE: STEP 1

Provisioning a cluster:

- eksctl create cluster --name=polito-cluster --nodes=1 --node-type=t2.small

- Uses a single t2.small Amazon Elastic Compute Cloud (EC2) instance as the worker nodekubectl get nodes
- kubectl get nodes

DEMO APP

CODICE FISCALE SERVICE: STEP 2

Deploying microservice:

- The first step of deploying to Kubernetes is to build your microservices and containerize them.

```
FROM adoptopenjdk/openjdk11:jdk-11.0.2.9-slim
WORKDIR /opt
ENV PORT 8080
EXPOSE 8080
COPY *.jar /opt/fiscalcode-0.0.1-SNAPSHOT.jar
ENTRYPOINT exec java $JAVA_OPTS -jar fiscalcode-0.0.1-SNAPSHOT.jar
```

docker build –t fiscalcode .

DEMO APP

CODICE FISCALE SERVICE: STEP 3

Pushing the image to container registry:

1. Authenticate docker client to your ECR

   PASSWORD=$(*aws ecr get-login-password)*

2. ACCOUNT=$(aws sts get-caller-identity --output text --query "Account“)
3. docker login -u AWS -p $PASSWORD https://$ACCOUNT.dkr.ecr.eu-west-1.amazonaws.com
4. aws ecr create-repository --repository-name fiscalcode

DEMO APP

CODICE FISCALE SERVICE: STEP 3

Pushing the image to container registry:

1. docker tag fiscalcode [fiscalcode-repository-uri]:0.0.1SNAPSHOT
2. docker push [fiscalcode-repository-uri]:1.0-SNAPSHOT

DEMO APP

CODICE FISCALE SERVICE: STEP 4

Deploying the microservices:

- Run the following commands to deploy the resources as defined in kube directory:

  kubectl apply -f kube

- SECURITY_GROUP=$(aws ec2 describe-security-groups --filters Name=group-name,Values="*eksctl-guide-cluster-nodegroup*" --query "SecurityGroups[*].{Name:GroupName,ID:GroupId}“)

DEMO APP

CODICE FISCALE SERVICE: STEP 4

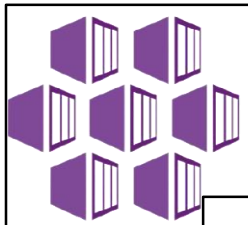
Making requests to the microservices:

- SECURITY_GROUP=$(aws ec2 describe-security-groups --filters Name=group-name,Values="*eksctl-guide-cluster-nodegroup*" --query "SecurityGroups[*].{Name:GroupName,ID:GroupId}“)
- aws ec2 authorize-security-group-ingress --protocol tcp --port 32250 --group-id $SECURITY_GROUP --cidr 0.0.0.0/0
- kubectl get nodes -o wide

REGISTRATION SERVICE

DEMO APP

REGISTRATION SERVICE



POD

Kotlin + spring boot

Registration service

mongo DB

External Registry

DEMO APP

REGISTRATION SERVICE: STEP 1

Create Cluster:

```
api_fingerprint = ""
api_private_key_path = ""
region = "eu-frankfurt-1"
tenancy_id = "ocid1.tenancy.oc1..aaaaaaaarbhtwso7rak7heaqbuoxoxjkhq4hc5tafzaaw2eohtmreb3ecwpa"
user_id = ""
compartment_id = "ocid1.compartment.oc1..aaaaaaaaa3h3rbfbu7dyrnypci7zjkdssfsxrsqrqjvpzkw464mq36fnh45q«
cluster_name = "oke"
vcn_cidr = "10.0.0.0/16"
```

DEMO APP

REGISTRATION SERVICE: STEP 1

Create Cluster:

```
ssh_private_key_path = "[...]/polito/bastion_host/key"
ssh_public_key_path = "[...]/polito/bastion_host/pub"
operator_instance_principal = true
environment =  "labname"
kubernetes_version = "v1.18.10" # faremo poi upgrade "v1.19.7"
np1 = {shape="VM.Standard.E2.1",node_pool_size=1}
np2 = {shape="VM.Standard.E2.1",node_pool_size=1}
node_pools_to_drain = [ "np1", "np2" ]
metricserver_enabled = true
```

DEMO APP

REGISTRATION SERVICE: STEP 2

Deploying microservice:

- The first step of deploying to Kubernetes is to build your microservices and containerize them.

```
FROM adoptopenjdk/openjdk11:jdk-11.0.2.9-slim
WORKDIR /opt
ENV PORT 8081
EXPOSE 8081
COPY *.jar /opt/registration-0.0.1-SNAPSHOT.jar
ENTRYPOINT exec java $JAVA_OPTS -jar registration-0.0.1-SNAPSHOT.jar
```

docker build –t register .

DEMO APP

REGISTRATION SERVICE: STEP 3

Pushing the image to container registry:

- Authenticate docker client to your docker hub

  docker login

- docker tag register mnakhtar247/register:0.0.1- SNAPSHOT
- docker push mnakhtar247/register:0.0.1-SNAPSHOT

DEMO APP

CODICE FISCALE SERVICE: STEP 4

Deploying the microservices:

- To access cluster locally
- oci ce cluster create-kubeconfig --cluster-id [cluster-id] --file kubeconfig --region eu-frankfurt-1 --token-version 2.0.0 --kube-endpoint PUBLIC_ENDPOINT
- Run the following commands to deploy the resources as defined in kube directory:

```
kubectl --kubeconfig kubeconfig apply -f kube
```

DEMO APP

CODICE FISCALE SERVICE: STEP 4

Making requests to the microservices:

- kubectl --kubeconfig kubeconfig get nodes
- kubectl --kubeconfig kubeconfig get pods –watch
- kubectl --kubeconfig kubeconfig get svc

USER APP SERVICE

DEMO APP

USERAPP SERVICE

POD

Registration service

External Registry

DEMO APP

REGISTRATION SERVICE: STEP 1

Provisioning a cluster:

az aks create -g clud-akhtar-tesina -n polito-cluster

- Running this command creates an AKS cluster that is called ***polito-cluster*** with the resource group ***clud-akhtar-tesina***
- By default, three nodes are assigned to the node pool
- To add nodes manually option *--node-count -c* can be added in the above command

DEMO APP

USER APP SERVICE: STEP 2

Deploying microservice:

- The first step of deploying to Kubernetes is to build your microservices and containerize them.

```
FROM adoptopenjdk/openjdk11:jdk-11.0.2.9-slim
WORKDIR /opt
ENV PORT 8080
EXPOSE 8080
COPY *.jar /opt/user_app-0.0.1-SNAPSHOT.jar
ENTRYPOINT exec java $JAVA_OPTS -jar user_app-0.0.1-SNAPSHOT.jar
```

docker build –t userapp .

DEMO APP

USER APP SERVICE: STEP 3

Pushing the image to container registry:

- Authenticate docker client to your docker hub

  docker login

- docker tag userapp mnakhtar247/userapp:0.0.1-SNAPSHOT
- docker push mnakhtar247/userapp:0.0.1-SNAPSHOT

DEMO APP

USER APP SERVICE: STEP 4

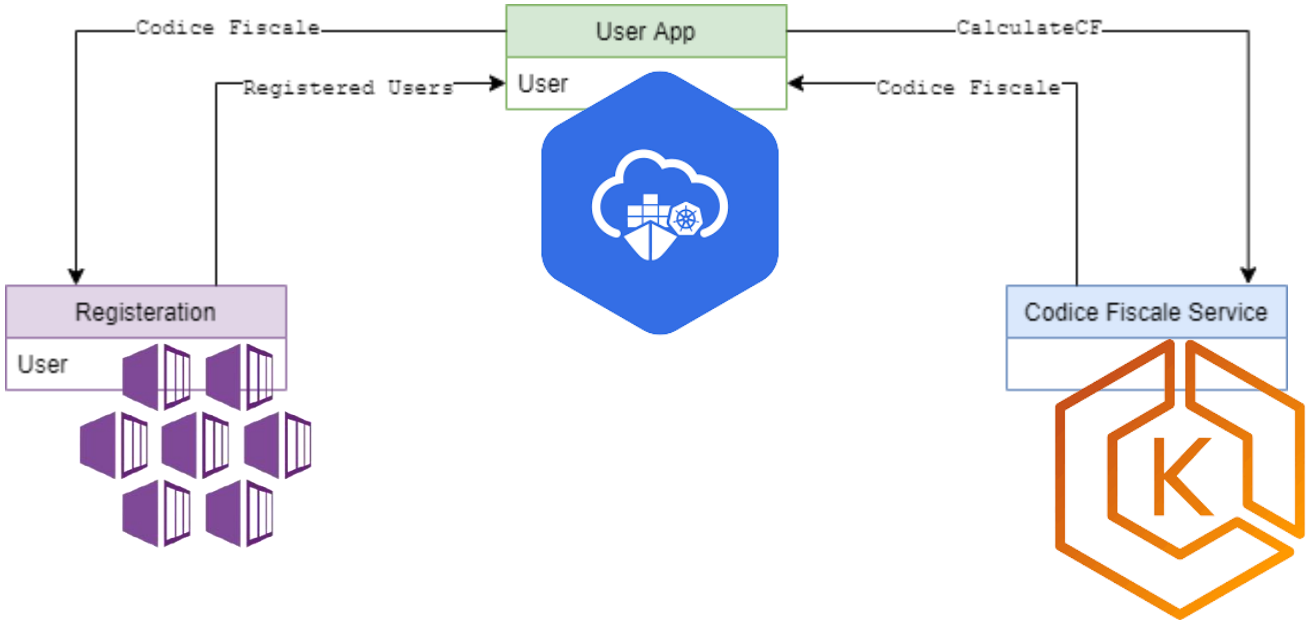
Deploying the microservices:

- az aks get-credentials -g clud-akhtar-tesina -n polito-cluster
- Run the following commands to deploy the resources as defined in kube directory:

  kubectl apply -f kube

- *kubectl get nodes*
- kubectl get service

DEMO APPLICATION



THANK YOU

www.reply.com

REPLY