Day-1

Why Microservices

- Developing web applications from early 2000s

  Process

  - Waterfall

  - Agile

- What is the burning need to consider new architecture

  - Scalability

  - Loosely coupled

  - Independency deployment.

  ad

SDLC overview

- Requirement analysis
- Design
  - Language (Java/Kotlin/GoLang/Python)
  - Framework (Spring/Hibernate/Maven/tomcat)
  - UML (Class/Sequence/Deployment/ER)
- Development
- QA
- Deployment

- More features
- More code
- More Dev/QA/Dev-ops
- More complex
- Features can get obsolete, but the code is not reflecting
- Release deadlines
- Introduction of bugs
- Leads to regression
- Impact analysis becomes challenges
- Agile in process but whether agile in release

- Reaching the features faster to your customers
- Scalability
- Global reach
  - Tally
  - Zoho/Quickbooks/Uber/Amazon

Monolith style of building applications

- You need to constantly innovate and have a competitive edge
- React faster to the market dynamics
- Hit the plateu
- Slower release cycles

When not to go for Microservice

- Internal application
- CRUD style of product
- Application develpment is manageble
  - small team size
  - smalled code base

Changes needed to move to microservices

- Emphasis on Automation
  - No manual intervention
- Devops process
- No single point of failure
    - Building redundancy
  - Leverage the managed services which can be a single point of failure

    ex: databases, message brokers, api-gateway
  - Platform provides the availability and durability guarentees

ex - API gateway, Databases, Message brokers

Difference between Monolith and Microservice

Monolith

- Design the system considering the system will never fail
- Availiablity - 0 / 1

Microservices

- Design the system considering that components will fail
- Availability - 0 .. 1

Implement the challenges in building Microservices

- Application layer - V-1
  - Netflix OSS components
  - home groon solutions
    - Eureka - Service discovery

server - Configuration management

- Zuul - Availibility
- Ribbon - Client side load balancing
- Hystrix - Resiliency

Features

- Concerns are managed at the application layer
- It is primarily built for JVM
- Deploy this solution on OnPrem/Hybrid/Cloud
- Application dev team shoud manage the availiabity/scalability/fault tolerance
- Applications are microservice aware
- Do not want to have the overhead of Operations

- Containers and Container Orchestration solutions (K8s/Openshift/Docker-swarm/Vmware-Tanzu/EKS/AKS)

Platform layer Generation-2

- All the chanlleges are pushed to the infrastructure/platform
- Applications are light weight
- Applicaions are not microservice aware
- Resource optimized
- Polyglot microservices
- Dedicated Dev-ops team to manage the cluster

- Serverless

Cloud offering Generation-3

- Serverless offering from the Cloud Partner1
  - Paramerter store, Secret manager, API gateway, Load-Balancer, Dynamodb
- Highly available
- More secure
- Pay only for the resource utilization - Cost effective
- Vendor lockin
- No/Min dev-ops needed to manage the resources

Challenges

- Decomposition
  - Monolith to Microserice
  - Microservice

Solution

- By Business capability
- Applying Domain driven design techniquies
- Identify all the sub-domains
- Identify the core sub-domain

- Set the bounded context

From the domain, split into sub-domains

- The sub-domain, which is core to your domain is refereed to as core- subdomain
- Competitive edge with the competitors
- Innovate/ship/deliver faster to the customers
- Core-subdomain should always be lean
- Domain experts/ Core-Dev team should be part of the core-domain
- Core-subdomain
- sub-domain
  - outsource
  - home groon
    - off the shelf solution

Project setup

1. Clone the repository
   - git clone https://gitlab.com/12-12-vmware-microservices/orders-microservice.git
   - Open the STS/IntelliJ Idea
   - Import the porject as "existing Maven project"
   - To pull the changes to the project - git pull origin master

2. Access the REST API using

   http://localhost:8222/api/v1/orders

3. To package the application

   mvn clean package -DskipTests

   java -jar <jar-file>.jar

Introducing Docker

Drawbacks of Virtual machines

1. The size of the VM's will be in GB's
2. The bootup time will be in minutes
3. Softwares, commands and applications on the kernel should be constantly patched

Docker is a self contained module consisting of

- Kernel
- Platform
- application binaries
- configuration

Docker images

- Read only
- Immutable
- Made up of layers
- Contains the OS as the base image
- Acts as a template to create docker containers

Docker containers

- writable layer on top of image

Docker properties

- Size of the images will be in MB's
- Memory footprint of the containers is very small
- Bootup time of the containers is in seconds
- Better resource utilization
- Ideal for running microservices workload

Link to setup docker

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html

Docker commands

docker info

docker images

docker container ls --all

docker pull hello-world

docker images

Default docker registry - hub.docker.com

docker container run hello-world

docker container ls

docker container ls --all

docker container run --name my-first-container hello-world

docker container ls --all

docker container run -d -p 80:80 nginx

docker container ls

docker logs <container-id>

curl http://localhost

docker container exec -it <container-id> /bin/bash

cd /usr/share/nginx/html

echo hello-world > index.html

exit

imperative commands

docker container stop <container-id>

docker container start <container-id>

docker container restart <container-id>

docker container stop <container-id>

docker container rm <container-id>

Dockerfile

- Set of instructions using docker commands
- The name of the file is Dockerfile (without any extension)
- The Dockerfile contains the instructions to create a docker image
- The Dockerfile can be part of the code repository
- The Dockerfile can be versioned controlled and goes through change management process

Link for the repository - https://gitlab.com/31-10-jpmc-k8s/orders-microservice.git

Docker lab - https://gitlab.com/classpath-docker/docker-lab

Install git on the server

```
sudo yum install -y git
git version
git clone https://gitlab.com/10-10-vmware-microservices/orders-microservices
cd order-microservices
```

Command to run the docker file to create the docker image

```
docker build -t orders-microservice .
```

Docker repo - https://gitlab.com/classpath-docker/docker-lab

To push the image to the docker-hub

```
docker login
```

tag the source image to the format that can be uploaded to docker-hub

```
docker tag orders-microservice classpathio/order-microservice

docker push classpathio/order-microservice
```

To run the order-microservice docker container

```
docker container run -d -p 8222:8222 classpathio/order-microservice
```

To log into the container

docker container exec -it <container-id> /bin/bash

javac

java -version

To exit from the container, run the exit command

Setup the CI to automate the process of image creation

Mordern tools like Github/Gitlab have the serverless offering to build the image pipeline as a code

- write the script to automate the build process and trigger on commit
- Pipeline will be committed to the repository for change management process

Github - Actions

Gitlab - Pipeline .gitlab-ci.yml

AWS - Codebuild buildspec.yml

Orders microservice

- http://localhost:8222/api/v1/orders
- http://localhost:8222/api/v1/orders?page=1&size=15&order=desc&field=price

H2-consolw

- http://localhost:8222/h2-console

Day 2

Install kubectl client

https://kubernetes.io/docs/tasks/tools/

To setup a K8s cluster on local machine

- Minikube - https://minikube.sigs.k8s.io/docs/start/
- Docker for Desktop - single not K8s cluster
- Digital ocean K8s cluster - 200$ credit for 2 months
- EKS/AKS - cloud providers

To connect to the K8s cluster

- Create a directory called .kube inside home directory
  - Mac/Linux ~/.kube
  - Windows C:\Users\<username>\.kube

To install AWS cli

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

K8s architecture

Components of Control plane

- API server
  - controller which accepts the request
  - client will make a POST request with the manifest
- etcd
  - distributed database to store the manifests
- Scheduler
  - Identify the worker node to orchestrate the contatiner
  - Assigns the manifests to the worker node
- Controllers

Components of Data plane

- Worker nodes
  - kubelet agent
  - Kubelet agent will download the docker image and run the container

Clients

- communicate with the API server
- clients securely connects with the API server by using the auth token

Options to deploy K8s cluster
- You manage both the control plane and the data plane
  - ex - On Prem
- Platform manages the control plane and you manage the data plane
  - you do not have access to the control plane
  - you need to manage the worker nodes
- Serverless
  - Both the control plane and data plane are managed by platform
    - EKS with Fargate (AWS)

K8s commands

generic syntax

kubectl get <resource>

kubectl describe <resource> resource-name

kubectl delete <resource> resuorce-name

Pods -> pods, pod, po

Replicaset -> replicaset, rs

Namespace -> namespace, ns

Service -> service, svc

ConfigMaps -> configmaps, cm

K8s documentation - https://kubernetes.io

Namespace

Create Namespace

- create namespace - kubectl create ns <yourname>-ns

- list the namespace - kubectl get ns

- set the namespace as the default namespace - kubectl config set-context --current --namespace=pradeep-ns

POD

Deploy the Pods

- create the yml POD definition

- create the pod - create the pod - kubectl apply -f K8s-order-microservice.yml

- kubectl logs order-microservice

- view the pod - kubectl get pods

- describe pod - kubectl describe po order-microservice

- view the pod in all namespaces - kubectl get pods --all-namespaces

- view the logs - kubectl logs order-microservice

- describe the pod - kubectl describe po order-microservice

  - useful for troubleshooting

- log into the pod

  - kubectl exec -it order-microservice -- /bin/bash

  - curl http://localhost:8222/api/v1/orders

  - cd /app/lib

  - ls

  - exit - exit from the container

- delete the pod - kubectl delete pod order-microservice

- delete all the pods - kubectl delete pod --all

- Once the pod is deleted, the pod cannot be restarted

Steps

- kubectl apply command - A POST request is sent to the API-server with the payload

- API server validates the yaml/json payload

- API server then adds the default values

- API server then stores the manifests files inside a distributed database called etcd - (distributed database)

- The scheduler picks up the payload and identifies the worker node to which the resource is scheduled

- API server will then instruct the worker node to manage the container

- The worker node has an agent called kubelet

- The kubelet downloads the image from the docker registry and starts and manage the container

Labels:

- To group the resources

- Filter the resources

- consists of key and value pairs

- key and value are both strings

- key should be unique and value can be duplicate

- labels can be attached to any K8s resource

- Labels can be created while creating the pod in the pod yaml definition

- Labels can be created after the pod has been created

- Labels can be updated and deleted

attach a label to the pod

- kubectl label pod order-microservice env=dev app=order-microservice version=1.0.0 tier=backend

display the label - kubectl get pods --show-labels

update the label - kubectl label pod order-microservice env=prod --overwrite

delete the label - kubectl label pod order-microservice version-

filter the pods with labels - kubectl get pods -l key=value

To view the manifest definition stored in the etcd

- kubectl get ns pradeep-ns -o yaml

- kubectl get pod order-microservice -o yaml

To fetch the worker nodes

- kubectl get nodes
- kubectl get nodes --show-labels

Replica-Set

- It is a controller
- matches the desired number of pods with the actual number of pods
- Replica-set manages the creation of pods
- short notation is rs

Replica-set Lab

kubectl get rs

kubectl get pods

kubectl get pods --show-labels

kubectl delete po --all

kubectl get rs

kubectl get pods

kubectl delete rs order-microservice-rs

scale the number of pods - kubectl scale rs order-microservice-rs --replicas=3

kubectl get pods --show-labels

kubectl label po <pod-name> app=order-microservice-debug --overwrite

kubectl get pods

kubectl label po <pod-name> app=order-microservice

kubectl get pods

Service

- Server side loadbalancer and distributes the traffic to available pods
- Service is managed by K8s and is highly available
- It provides fixed IP/constant IP addresses within a K8s cluster
- The cluster IP is assigned by a DNS server present inside the K8s cluster
- The cluster IP is resolvable within the cluster only and not accessible outisde the cluster

- The service to service communication should happen via the service
- There is no relationship between service and Replica-set

Lab:

- kubectl apply -f k8s-order-microservice.yml
- kubectl get svc
- kubectl describe svc order-microservice-svc
- Log into one of the pod

Inventory microservice

clone the project git clone https://gitlab.com/10-10-vmware-microservices/inventory-microservice

https://gitlab.com/14-11-synechron-microservices/inventory-icroservice.git

import as existing maven project

start the inventory service

Service to service communication

POST -> http://localhost:8222/api/v1/orders/

Body ->


```
{
  "price": 2000.67,
  "customerName": "Vinay",
  "email": "Vinay@gutmann.net",
  "lineItems": [
    {
    "name": "Aerodynamic Aluminum Shirt",
    "qty": 3,
    "price": 540.3
    }
  ]
```

```
}
```

Lab for service to service communication:

deploy the inventory microservice manifests

```
kubectl apply -f  kubectl apply -f k8s-inventory-microservice.yml
kubectl get svc
kubectl get rs
kubectl get pods
```

delete the pods of order-microservice pods

```
kubectl delete po --all
```

verify if all the pods are available

```
kubectl get pods
```

login to one of the order-microservice pod

```
kubectl exec -it <pod-name> -- /bin/bash
curl --location --request POST 'http://localhost:8222/api/v1/orders' \
--header 'Content-Type: application/json' \
--data-raw '{
    "name": "amit",
    "email": "amit@gmail.com",
    "price": 5000,
    "orderDate": "2022-11-11",
    "lineItems": [
        {
            "name": "Heavy Duty Copper Coat",
            "qty": 2,
            "price": 421.50
        }
    ]
}'
```

exit from the pod

check the logs

kubectl logs <pod-name>

kubectl logs -f --selector app=order-microservice

ConfigMaps

- externalizing the application configuration
- key-value pair
- key should be unique
- value can be
  - string
  - contents of a file
- kubectl create cm <config-map-name> --from-literal=key=value --from-literal=key=value
- kubectl create cm <config-map-name> --from-file=<folder-name>

Lab:

- clone the manifest repository - git clone https://gitlab.com/10-10-vmware-microservices/manifests.git
- kubectl create cm order-microservice-cm --from-file=manifests
- kubectl get cm order-microservice-cm
- kubectl describe cm order-microservice-cm
- delete all the existing pods - kubectl delete po --all
- login to one of the pod - kubectl exec -it <pod-name> -- /bin/bash
- exit from the pod and check the application logs - kubectl logs <pod-name>
- create the replica-sets using kubectl apply -f K8s-order-microservices.yml
- login to one of the pod - kubectl exec -it <pod-name> -- /bin/bash
- navigate to /app/config directory
- list the files and you should be seeing the application configuration files inside the directory
- create the db-credentials config-maps - kubectl create cm db-credentials --from-literal=username=sa --from-literal=password=welcome

- list the config maps - kubectl get cm

- delete all the pods - kubectl delete po --all

- fetch all the pods - kubectl get po

- login to one of the pod - kubectl exec -it <pod-name> -- /bin/bash

- run the env command to verify the SPRING_DATASOURCE_USERNAME and SPRING_DATASOURCE_PASSWORD environment variable present

- exit from the pod - exit

- run the logs command - kubectl logs <pod-name>

- verify that the pod is running in the "qa" mode

Secret

- Just like configmaps

- used to store key value pairs

- Value is stored in base64 encoded format

- Create a secret

kubectl create secret generic db-credentials --from-literal=password=welcome

Labs for working with Service account

- Make the repository private in hub.docker.com

- Delete all the pods : kubectl get podskubectl delete po --all

- New pods will come up because they are managed by Replica-Set

- The Kubelet inside the new pods will not be able to pull the docker image from the registry

Pulling an image from the private repository

- https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/

- kubectl create secret docker-registry docker-credentials --docker-server=https://index.docker.io/v1/ --docker-username=classpathio --docker-password=Welcome44 --docker-email=pradeep@classpath.io

- kubectl get secret docker-credentials -o yaml > docker-credentials.yaml

- Inject the secret in the imagePullSecret in the pod's container definition

- Delete the replica-sets

- Apply the replica-sets changes

- The pods should now be able to pull the images from the private docker registry

Injecting the image pull secrets with the help of service account

- Fetch the service account

  kubectl get sa default -o yaml > sa.yaml

Resiliency

- Need to ensure that a failing service does not bring down the entire application
- Give the failing service time to recover
- Do not overwhelm by calling a failing service
- To implement self healing application
- Prevent from downstream API's from failing
- Only exceptions relation to IO should be treated as failures

Options

- Provide a fallback mechanism
- retry (should be limited)

Implementation:

- Add the Resiliency4j library to the pom.xml
- Add the resiliency4j configuration
- monitor the status of circuit breaker using the actuator endpoint
    - http://localhost:8222/actuator/health

POST-request - http://localhost:8222/api/v1/orders/

```
{
   "name": "Praveen",
   "email": "praveen@gmail.com",
   "price": 5000,
   "orderDate": "2022-11-11",
   "lineItems": [
      {
```

```
        "name": "Heavy Duty Copper Coat",
        "qty": 2,
        "price": 421.50
      }
   ]
}
```


Day-3

- Resiliency with K8s
  - Applications should expose the health endpoints
  - K8s can be configured to call the health endpoints
  - K8s can be configured to execute the below probes
    - execute a script
      - should return a non-zero exit code
    - execute a command
    - execute a HTTP call
      - To be success, the response should be 2xx series
      - Liveness probes
      - Readiness probes

Considerations for configuring the health endpoints from the application layer

- It should be a GET request
- It should not be a protected resource (No Authentication and Authorization)
- The health endpoints should not take more than few milliseconds to response

Liveness probes:

- The K8s will call the liveness probes with the configured frequency
- If the liveness probe fails, then the pod will be restarted

Readiness probes:

- The K8s will call the readiness probes with the configured frequency
    - If the readiness probe fails, then the requests will not be sent to the pod

Implementing the liveness and readiness probes in Spring Boot application

POST -> http://localhost:8222/state/liveness

Body -> none

POST -> http://localhost:8222/state/readiness

Body -> none

Lab:

Liveness probe:

- apply the changes - kubectl apply -f k8s-order-microservice.yml
- kubectl get pods
- login to one of the pod - kubectl exec -it <pod-name> -- /bin/bash
- update the liveness probes state - curl -XPOST http://localhost:8222/state/liveness
- verify that the liveness probe is down - curl http://localhost:8222/actuator/health/liveness
- exit from the container
- fetch the pods - kubectl get pods
- verify that the pod restarts and the restart count is incremented

Readiness probe:

- apply the changes - kubectl apply -f k8s-order-microservice.yml
- kubectl get pods
- login to one of the pod - kubectl exec -it <pod-name> -- /bin/bash
- update the readiness probes state - curl -XPOST http://localhost:8222/state/readiness
- verify that the readiness probe is down - curl http://localhost:8222/actuator/health/readiness
- exit from the container
- fetch the pods - kubectl get pods
- verify that the pod state is Not-Ready
- To make the pod to accept the traffic

- login to one of the pod - kubectl exec -it <pod-name> -- /bin/bash
- update the readiness probes state - curl -XPOST http://localhost:8222/state/readiness
- verify that the readiness probe is up - curl http://localhost:8222/actuator/health/readiness
- exit from the container
- fetch the pods - kubectl get pods
- verify that the pod state is changed to Ready

Security

- Security should be implemented using Defence in depth strategy

  Levels of security

  - Data
    - Data should be encrypted at rest
    - Use strong encryption alogorithm - AES-256
    - Do not store sensitive information in the log files
    - Use one-way hash functions to store sensitive information in the database
  - System
    - Rule of minimum previligous
    - Min access to all the resources
    - No unnecessary applications/softwares
    - Continous patching and updates for vulnarability
    - Vulnarability scanning of docker images, VMs etc
    - No root access to the machines
  - Network
    - Only the allowed ports should be open
    - Seggregate private subnets and public subnets
    - Setup firewalls, security groups
    - Encryption at transit, TLS-1.2, HTTPS, certificates
    - Ratelimitting, timelimitting
  - Application

- Authentication
  - Whether you are the same person, whom you claim to be
  - HTTP - 401 - UnAuthorized
- Authorization
  - Do you have necessary priviligous to acees the resources
  - HTTP - 403 - Forbidden
- Even if one of the layers is compromised, the next line of defence should be even more stonger
- The strength of a chain, depends on the strength of its weakest link
- If there is a vulnaribility, then the vulnaribility WILL be exploited by Advaisary

OAuth 2.0

- Delegation based Security framework
- The PRINCIPAL will DELEGATE PART_OF_RESPOSIBILITY to a TRUSTED_APPLICATION to PERFORM_ACTION on his BEHALF

Vocabulary from Security domain

- Anonymous user - An entity who is not yet authenicated
- Entity
  - User
  - Machine
  - Program
  - Service
- Principal
  - Authenticated entity is referred to as Principal

OAuth 2.0

- It is a framework
- Grant flows depending on the type of client
- Different types of clients are supported
  - Public trusted clients - (End users)

- Backend application - Deploy your applications on servers

  ex: Java, Python, Nodejs

  - Store confidential information

  - Back channel

  - Secure

- Front end applications - Mobile apps, SPA

  ex: Angular/React/JS

  - Front channels

  - Cannot store confidential information

  - insecure

- Private (No user, Machine to machine communication)

  - application/service - application/service communication

  - back channel

  - secure

Grant types

- Authorization code
  - Backend public client
- Proof of Key Exchange - PKCE
  - Front end public client
- Client Credentials
  - service to service communication
  - microservice to microservice communication

OAuth 2.0 actors

- Resource (Protected)
  - API's
  - Orders api
  - Inventory api
- Resource Owner

- End Users
- Client application
  - will be building a Spring Boot application (backend) - Auth code grant flow
- Authorization Server
  - Out of the box implementation - OKTA
  - Options
    Social
      Google
      Facebook
      Github
    IDP - Enterprise hosted solutions
      OKTA
       - Provides Identity and management solutions
       - Developer account is free to use
      WSO2
    KeyCloak - Open source implementation
    - Can be self hosted

OAuth2 Auth Code workflow

- https://developer.okta.com/docs/guides/implement-grant-type/authcode/main/

Step-0

- Client application registers with the Auth server
  Onboarding process
    input -> redirect url
          http://localhost:8555/authorization-code/callback
    output -> client-id, client-secret
          public confidential
    client-id - 0oa5zuggpixxr5aQ45d7
    client-secret(sensitive-information) V-dfWtYXjhMp0T5gbQAODxeOEoDgBatRTV7s3V8y

Step-1

- Auth server exposes a metadata url
  - Also referred as well known url
  - Public endpoint

    https://dev-7858070.okta.com/oauth2/default/.well-known/oauth-authorization-server

Details:

issuer: "https://dev-7858070.okta.com/oauth2/default"

authorization_endpoint: "https://dev-7858070.okta.com/oauth2/default/v1/authorize"

token_endpoint: "https://dev-7858070.okta.com/oauth2/default/v1/token"

registration_endpoint: "https://dev-7858070.okta.com/oauth2/v1/clients"

jwks_uri: "https://dev-7858070.okta.com/oauth2/default/v1/keys"

Anonymous user try to access the client application

Step-2

Redirect the anonymous user to the Authorize endpoint of the Auth server

https://dev-7858070.okta.com/oauth2/default/v1/authorize

Query parameters

client_id - 0oa6vf06wtsub6ob65d7

response_type - code

scope - openid

redirect_uri http://localhost:8555/callback/url

state - 975733f8-f1c6-4eeb-918b-9bd8f9d86194

Construct the url

https://dev-7858070.okta.com/oauth2/default/v1/authorize?client_id=0oa5zuggpixxr5aQ45d7&response_type=

code&redirect_uri=http%3A%2F%2Flocalhost%3A8555%2Fauthorization-code%2Fcallback&state=975733f8-f1c6-4eeb-918b-9bd8f9d86194

Step-3

- Auth server will authenticate the user
- If the authentication is successful, then seeks confirmation to authorize
- If Authorized, will return back the auth_code in the response

http://localhost:8555/authorization-code/callback?code=nOc1qWgb_6JPpAU-yyRZR6LrfCDO0kzItaZkTX46eNw&state=975733f8-f1c6-4eeb-918b-9bd8f9d86194

- auth code = WyjhBDF_YVHF_MUVuKWJBuer7gLgv4MVhBiPLwAMH-4

validity is 5 minutes

- auth code is not secure
- it is transimitted over an untrusted network

Step-4

- The client applications(Backend application) uses the auth code
- Exchanges the auth cod with the access token using the POST request with the token endpoing
  - POST method
  - Endpoint - token endpoint

    https://dev-7858070.okta.com/oauth2/default/v1/token
  - Body

    grant_type - authorization_code

    redirect_uri - http://localhost:8555/authorization-code/callback

    code - GBPenQJAEGZp5vj8wDcf0R5ipJEeVF7wt1a6WuVOT1A
  - Basic authentication

    client-id - 0oa5zuggpixxr5aQ45d7

    client-secret - V-dfWtYXjhMp0T5gbQAODxeOEoDgBatRTV7s3V8y

Response

{
"token_type": "Bearer",
"expires_in": 30000,

"access_token":
"eyJraWQiOiJGQkpPV0FxUEF1N3B1U25pSmQyYWoxNmNvNVh3aXA1OUxBeV9hRkk4ZFFJIiwiYWxnIjoiUlMyNTYifQ.eyJ2ZXIiOjEsImp0aSI6IkFULktrZ1NHYml2Q3dnSHY4NTN0V1d5YTd3U25jVTBKeW8zWDlzdGlaS3FPTk0iLCJpc3MiOiJodHRwczovL2Rldi03ODU4MDcwLm9rdGEuY29tL29hdXRoMi9kZWZhdWx0IiwiYXVkIjoiYXBpOi8vZGVmYXVsdCIsImlhdCI6MTY2ODU5NDIzOSwiZXhwIjoxNjY4NjI0MjM5LCJjaWQiOiIwb2E1enVnZ3BpeHhyNWFRNDVkNyIsInVpZCI6IjAwdTNteXk1c09sOVNEYnYzNWQ2Iiwic2NwIjpbInByb2ZpbGVfaW5mbyIsImRldmVsb3BlciJdLCJhdXRoX3RpbWUiOjE2Njg1OTQwNzgsInN1YiI6InByYWRlZXAua3VtYXI0NEBnbWFpbC5jb20iLCJncm91cHMiOlsiRXZlcnlvbmUiLCJzdXBlcl9hZG1pbnMiLCJhZG1pbnMiXX0.lxiILM_wKSsozObnumD_BhvyPWL0hRWNYG7EcJoUNbWmJkHyaAHNc9EkL05zfHoHawcg2EkuIurgtYGj1LFEApbvAPjBDHply0Hn-fz1P6dyeed47rQY1aowiDlxhu-VqLnAj1Y3oRXK1Xntah8OQmRAZSfWr1so4bWGiBl0qHCraK5z4BuIVBHyUCLMHYYBGTjJUGK-RwKMLDNN2C5Ds5OVFn1xHbqnpdNOL9eAQ06YZ3ctIS05xYNLVqft3fEpnOWihXZapKpGkfp4FSVBFaeuxOUgh2JD6Axchk-arzyhokEKcwX_BHEid-hek3MhG-0sPr3LWEIPg8UuV1lTtQ",

"scope": "profile_info developer"

}

JWT token

- Crytographically secure token
- Self contained token
- Not to store sensitive information
- It is a digital signature of user identiry
- short lived token
- Validate the token - https://jwt.io

Step-5

- The client application will send the token to the resource server in the Authorization header
- The resource server will decode the token
- Extract the claims
- Allow/Deny the resource access
- 403 in case of forbidden usecase

OAuth2-client-application

codebase - git clone https://gitlab.com/10-10-vmware-microservices/ekart-oauth2-client.git

https://gitlab.com/14-11-synechron-microservices/ekart-oauth2-client.git

dependency of oauth2-client

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

start the server - port 8555

Invoke the endpoint - http://localhost:8555/api/userinfo

OAuth2-resource server

Transactions

Event

- State change which is of business interests
- Immutable
- Past tense

To list the dependency graph - .\mvnw dependency:tree > dependency.txt

Kafka repository

- https://gitlab.com/classpath2

Kafka installation

- Install Java

```
sudo yum install java-11-amazon-corretto
```

cd /opt

- Download Kafka binary

```
sudo curl -O https://dlcdn.apache.org/kafka/3.3.1/kafka_2.13-3.3.1.tgz
sudo tar -xvf kafka_2.13-3.3.1.tgz
sudo mv kafka_2.13-3.3.1 kafka
```

- Edit the server.proerties inside the config directory

listeners = http://localhost:9092

advitised_listeners = http://host:9092

Kafka-client commands

Move to the bin directory go inside the bin folder of kafka

cd /opt cd /kafka cd /bin

- Create a topic

sh kafka-topics.sh --create --topic pradeep-topic --bootstrap-server 13.233.190.33:9092

- List all the topics

sh kafka-topics.sh --list --bootstrap-server 13.233.190.33:9092

Producer

- Push the message to the topic

sh kafka-console-producer.sh --topic pradeep-topic --bootstrap-server 13.233.190.33:9092

- Consume the message to the topic

sh kafka-console-consumer.sh --topic pradeep-topic --from-beginning --bootstrap-server 13.233.190.33:9092

Kafka-server commands

- Move to the bin directory

Start the zookeeper

sh zookeeper-server-start.sh -daemon ../config/zookeeper.properties

Start the Kafka server

sh kafka-server-start.sh -daemon ../config/server.properties

- To create the topic

sh bin/kafka-topics.sh --create --topic <topic-name> --bootstrap-server localhost:9092

- To list the topics

sh bin/kafka-topics.sh --list --bootstrap-server localhost:9092

Lab:

POST http://localhost:8222/api/v1/orders

```
{
  "price": 2000.67,
  "customerName": "your-name",
  "email": "Vinay@gutmann.net",
  "lineItems": [
    {
    "name": "Aerodynamic Aluminum Shirt",
    "qty": 3,
    "price": 540.3
    }
  ]
}
```

Auth Header -> Bearer token

eyJraWQiOiJGQkpPV0FxUEF1N3B1U25pSmQyYWoxNmNvNVh3aXA1OUxBeV9hRkk4ZFFJIiwiYWxnIjoiUlMyNTYifQ.eyJ2ZXIiOjEsImp0aSI6IkFULlJLTmlhMDBWSEg2Z0Rnc3FudVExNVdRbjVtNVJROGpMYnlwRS1FZlF4NUEiLCJpc3MiOiJodHRwczovL2Rldi03ODU4MDcwLm9rdGEuY29tL29hdXRoMi9kZWZhdWx0IiwiYXVkIjoiYXBpOi8vZGVmYXVsdCIsImlhdCI6MTY2NTU1NjkyMiwiZXhwIjoxNjY1NTg2OTIyLCJjaWQiOiIwb2E1enVnZ3BpeHhyNWFRNDVkNyIsInVpZCI6IjAwdTNteXk1c09sOVNEYnYzNWQ2Iiwic2NwIjpbInByb2ZpbGVfaW5mbyIsImRldmVsb3BlciJdLCJhdXRoX3RpbWUiOjE2NjU1NTY0NjUsInN1YiI6InByYWRlZXAua3VtYXI0NEBnbWFpbC5jb20iLCJncm91cHMiOlsiRXZlcnlvbmUiLCJzdXBlcl9hZG1pbnMiLCJhZG1pbnMiXX0.iVWNXbtymk2SUYtaCsillsGE7TF4BC-52MG_HHfDRat_epzk9CY21mVks79Vr7mZMh0jfxZ1HR1Wpi_c1gCNryCxF6pfJf31kOfLJE63arkBgXcrrL_y498Mkq5UbmTho7ORsE_2qHwtOGtyKy4IxEv0kKVW3nRyTijUB9nEaZMp3CwOrUPD6EyepB3MSTF6sOIKHZMY9FDxMsEztrY7eoKvU5CzGB6o_MoWE6SsJ4_Vtt8ljyMpJAee-TnoTH5Wy_foBBIuJ4PbbzpGoi38V7KtSnH_3RZWlZB2xo9tnYmvuzIYPnhWyv_ledTB9CdvdIcygim91JmibZ7bR84KGA

etc host file

Location

Windows - C:\Windows\System32\drivers\etc\hosts

Mac/Linux - /etc/hosts

64.225.86.83 pradeep.classpath.io

http://pradeep.classpath.io/orders/api/v1/orders

Drive link -
https://drive.google.com/drive/folders/1KJxmx9SVkxvLmXU76jI7_cAKPwXGLzl8?usp=sharing

Deployment

- kubectl apply -f k8s-order-microservice.yaml

- kubectl get deployment

- kubectl rollout status deployment <deployment-name>

- kubectl rollout history deployment <deployment-name>

- kubectl annotate deployment order-microservice-dep kubernetes.io/change-cause="image version 1.0.0"

- kubectl set image deployment/order-microservice-dep order-microservice-container=classpathio/order-microservice:68e14cf09295e4520ef69d055fc3634fdf030dd5

- kubectl get deployment

- kubectl get rs

- kubectl get pods

- kubectl rollout status deployment order-microservice-dep

- kubectl rollout pause deployment order-microservice-dep

- kubectl rollout resume deployment order-microservice-dep

EKS-cluster setup

IAM users to EKS binding

To set the kubeconfig with the terraform user

aws eks --region ap-south-1 update-kubeconfig --name eks --profile terraform

Create ClusterRole and ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: reader
rules:
  - apiGroups: ["*"]
    resources: ["deployments", "configmaps", "pods", "secrets", "services"]
    verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: reader
subjects:
- kind: Group
  name: reader
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name:  reader
  apiGroup: rbac.authorization.k8s.io
```

Create the policy for Read access

```
aws configure --profile developer
```

```
kubectl edit -n kube-system configmap/aws-auth
```

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
```

```
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapUsers: |
    - userarn: arn:aws:iam::831955480324:user/pradeep
      username: developer
      groups:
      - reader
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::831955480324:role/eks-node-group-general
      username: system:node:{{EC2PrivateDNSName}}
kind: ConfigMap
metadata:
name: aws-auth
  namespace: kube-system

kubectl config view --minify

kubectl auth can-i get pods
kubectl auth can-i create pods

Add admin group
Create EKS admin policy
{
    "Version": "2012-10-17",
    "Statement": [
```

```
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "eks:*",
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "iam:PassedToService": "eks.amazonaws.com"
            }
         }
      }
   ]
}
```

Create a role and assign the policy - eks-admin

Verify

aws iam get-role --profile terraform --role-name eks-admin

Establish the trust between the user and the role

Create a policy and attach to the admin user

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
```

```
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": "arn:aws:iam::831955480324:role/eks-admin"
        }
    ]
}
```

aws sts assume-role --role-arn arn:aws:iam::831955480324:role/eks-admin --role-session-name manager-session --profile pradeep

update the configmap with rolearn of eks-admin and attach the user to the system:masters group

Setting up K8s cluster

- Kubeadm - On Prem production cluster
- Minikube
  - single node k8s cluster
- Docker for desktop

K8s cluster

- Control plane
- Data plane

Managed K8s cluster

- DigitalOCean
- EKS
- AKS
- GKE