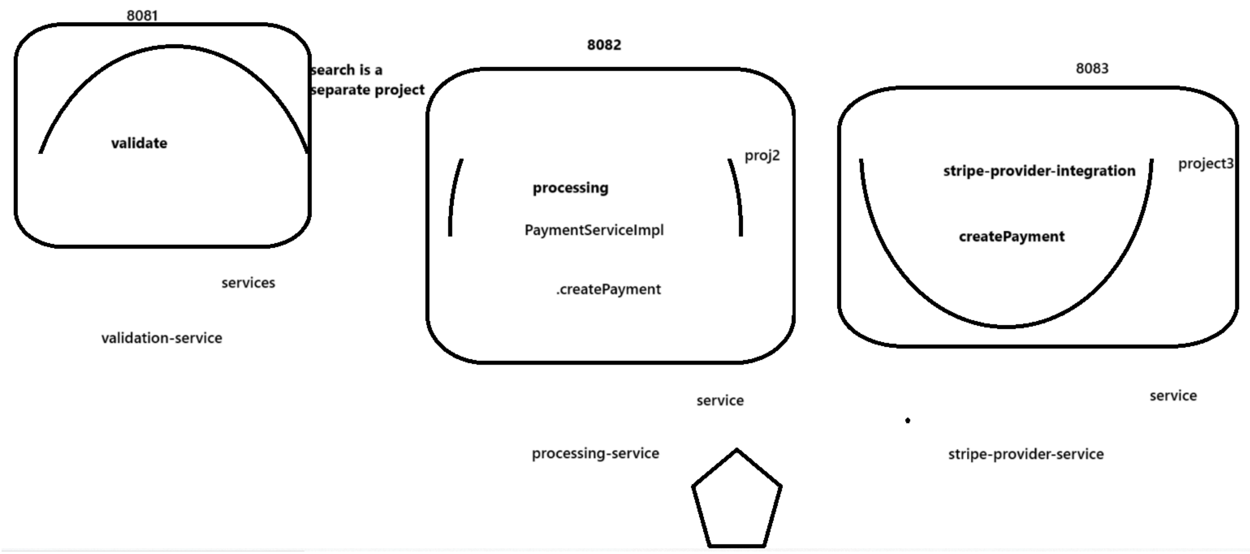
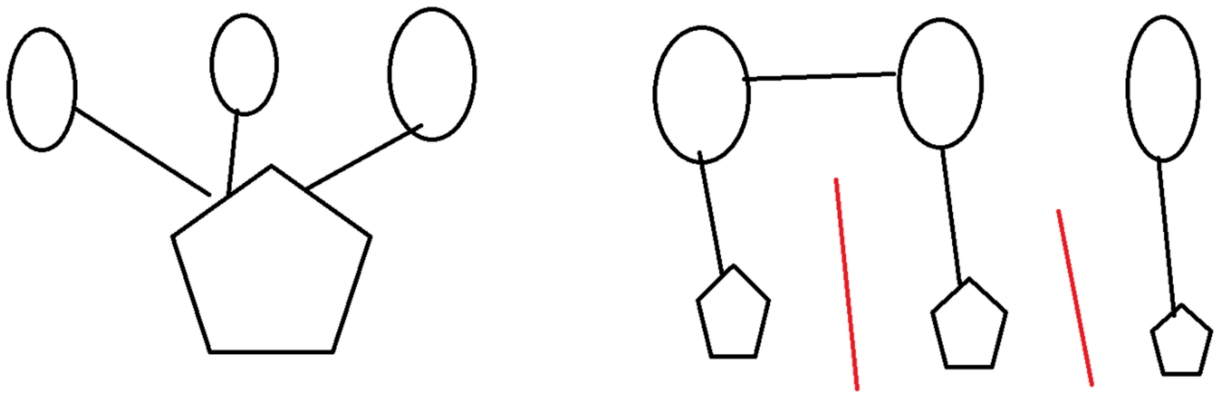
DIAGRAMS:

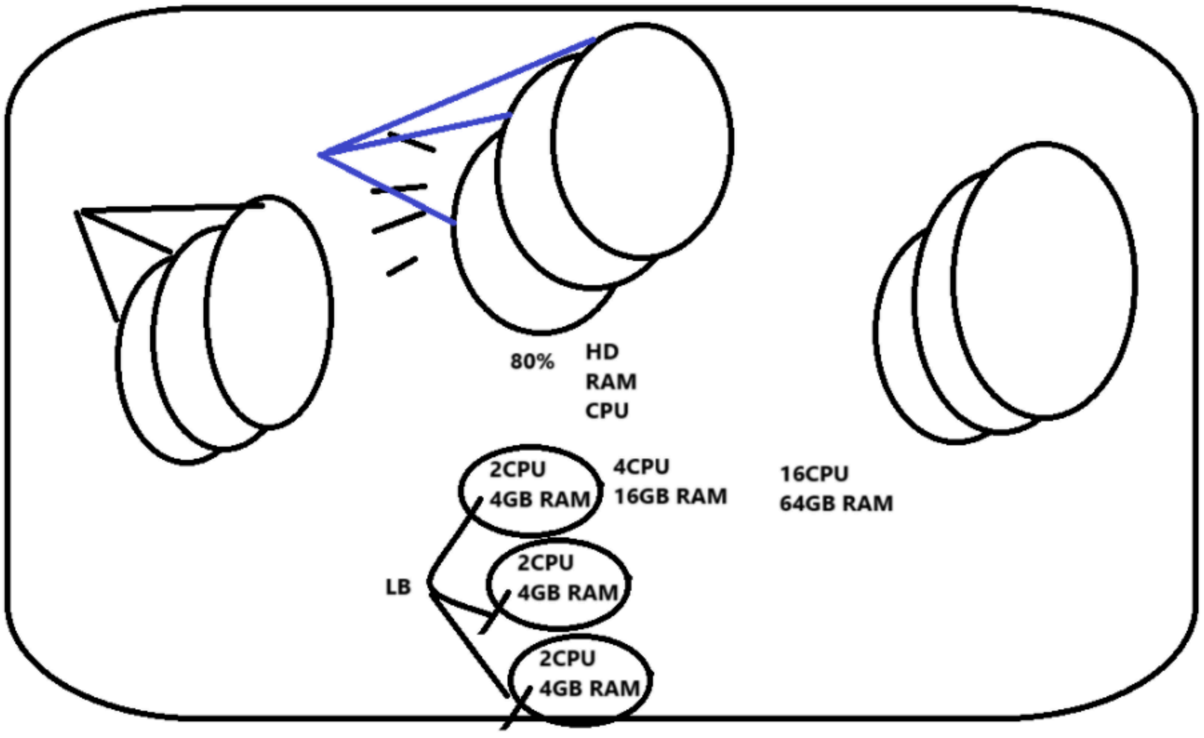
Breaking project into components & running as independent service



Each service to have its own DataBase



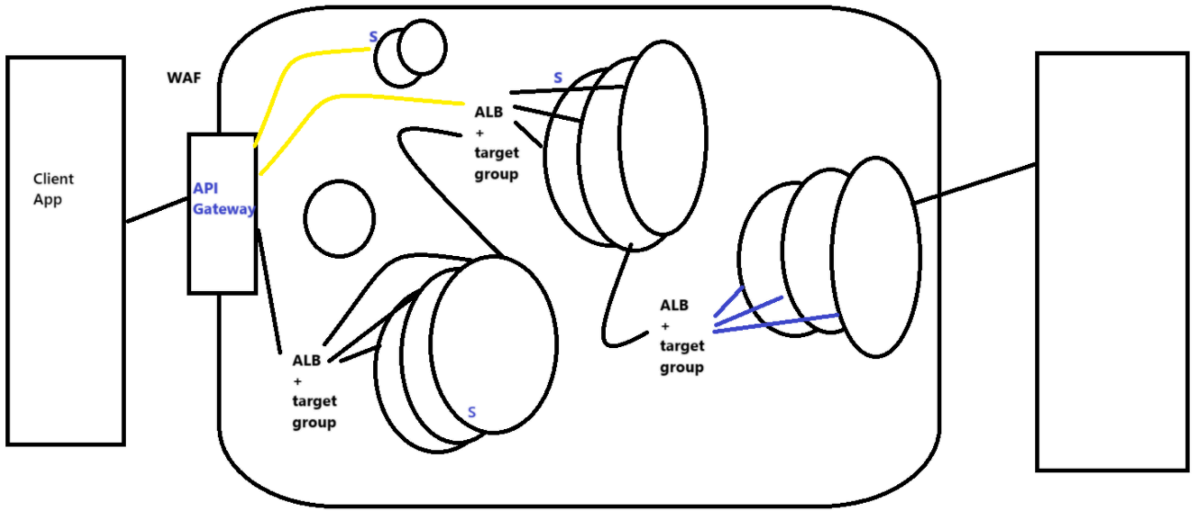
Services will be autoscaled & accessed behind load balancer



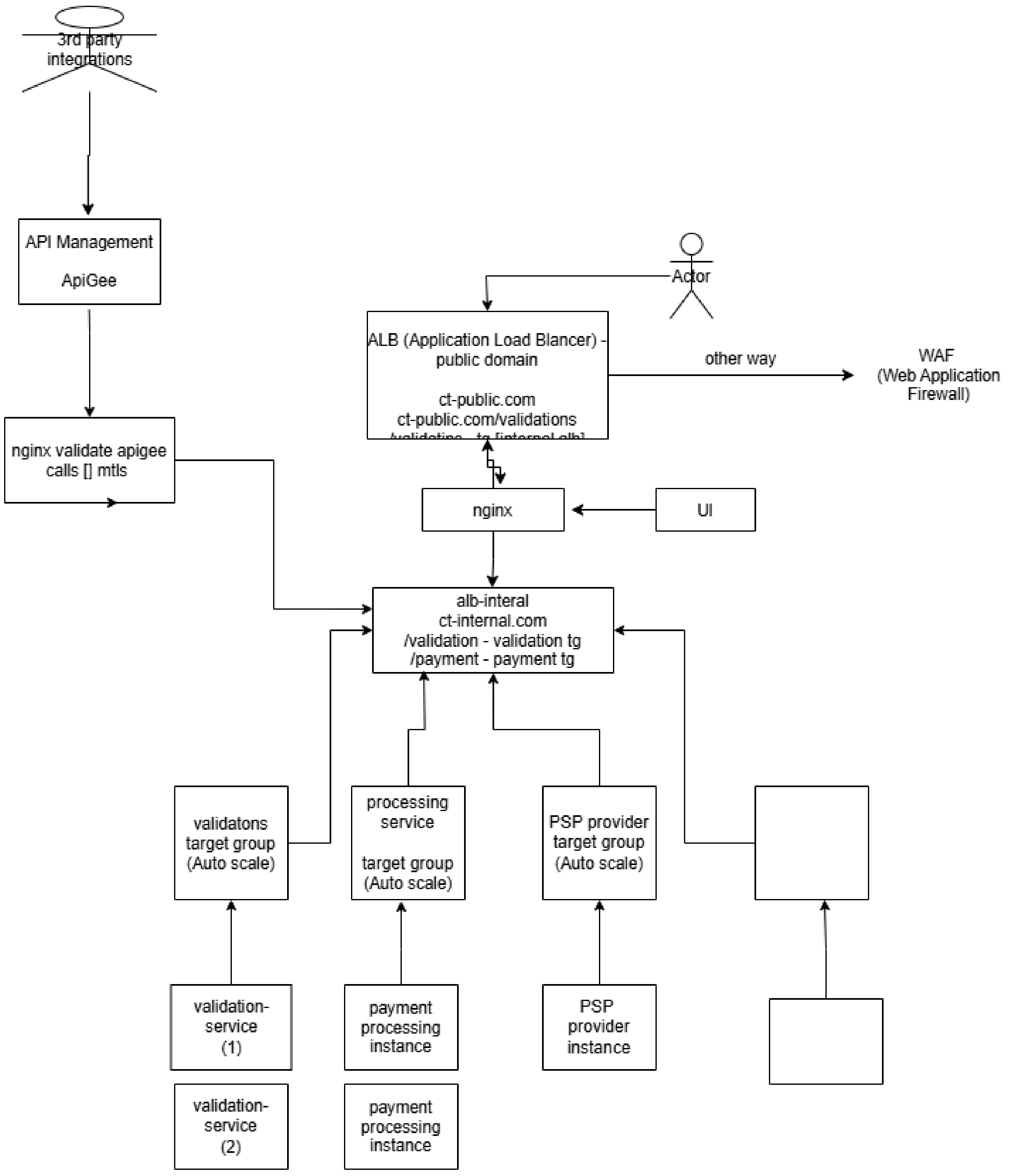
Flow of communication, Infra + Application

DNS

api.hulkhiretech.com ALB_public_IP



INFRA DIAGRAM


3rd party integrations
API Management
ApiGee
nginx validate apigee calls [] mtls
Actor
ALB (Application Load Blancer) - public domain
ct-public.com
ct-public.com/validations
other way
WAF (Web Application Firewall)
nginx
UI
alb-interal
ct-internal.com
/validation - validation tg
/payment - payment tg
validatons target group (Auto scale)
processing service
target group (Auto scale)
PSP provider target group (Auto scale)
validation-service (1)
validation-service (2)
payment processing instance
payment processing instance
PSP provider instance

LIVE NOTES

SDLC

- AGILE

Monolith

SOA

Microservices

Technical guidelines for building systems

Martin Fowler
9 principles of Microservices

Relate with the current project.

1. Componentization via services
2. Organized around business capabilities
3. Decentralized governance
4. Decentralized data management (When possible)
5. Infrastructure automation
6. Smart Endpoints and Dumb Pipes
7. Products, Not Projects
8. Design for Failure
9. Evolutionary Design

-----

1. Componentization via services

    Breakdown your project into smaller component / pieces / parts

    if entire project is in 1 piece, you an import other classes & call methods.
    breakdown into component, how can these piece talk to each other?

    Componentization via libraries

    Componentization via services

        each project will be coded separately
        Build separately

executed separately
it has its own memory
its own port
process id

Don't code entire project in one go, rather break it down into smaller component, & run each component independently as a service. its own memory & port.. complete separate build & deployment cycle.

java -jar min max

How do you break the system into smaller components???

2. Organized around business capabilities
    breakdown project based on business domain.
    group of related functionalities you put together.

validation-service
- is to validate each incoming payment request.
- Field validation & Business validations

- Spring Boot Security

processing-service:
- Core Payment processing
- Status Management

stripe-provider-service:
- is to integrate with StripePSP

3. Decentralized governance
    Team has the power to make decisions on how we can make the system better.
    you can choose the tool/tech, as per latest standards, to keep your project updated to current market trends...

4. Decentralized data management (When possible)
each microservice should have its own DB
only that microservice is allowed to talk to its own DB
other microservice cannot connect to other's DB.
if other microservice want to get something from cross DB, then connect to the responsible microservice & gets the work down.

5. Infrastructure automation
1 spring boot - end-to-end develop to golive actions..

1 git repo in bitbucket
master => integration => feature
setup feature in local machine
code in IDE
local testing
build & deploy in dev aws env
dev testing
merge to integration
build & deploy in qa aws env
bug
bug branch, fix bug, deploy in dev aws env
merge to integration
build & deploy in qa aws env
end of sprint
cut release branch,
build & deploy in uat
handling uat bugs
bug fix brach from release
test in local, dev, qa
merge code to release branch
build & deploy in uat env
merge to master branch
build & deploy in prod env.
cut hotfix branch
test in uat, qa
merge to master
build & deploy in prod env.
Jenkins pipeline

Devops - CI/CD

pool of (20 devops)
pool for DBA
pool of Architects

Auto Scaling

your service will have max memory (RAM), CPU, HD.. limits..
You has maximum concurrent processing capacity. if more than that comes, your service will not be able to respond.

Scale

Vertical Scale - update the same machine with bigger power
Horizontal scale - another instance of same service will be started.

Infra overview:
ALB Public facing(Application Load Balancer)
WAF rules
NginX - proxy
Internal ALB
service specific Target group (scaling)
Scalled up instance of our microservices.

API Gateway
1 time central security logic before passing requests to all internal services.
Entry to entire system.
Custom security logic
WAF rules can be applied
rate limiting applicable..

How to implement API Gateway.
1. Get API Gateway as a AWS service
2. You can write your own Spring Boot API Gateway application.

6. Smart Endpoints and Dumb Pipes:

SOA - Enterprise Bus

Light weight
RestAPI over HTTP
JSON RPC over HTTP

Messaging Brokers. Kafka, ActiveMQ, RabbitMQ, Amazon SQS...

1. RestCLient API calls from service1 to service2
    light weight compared to 2nd approach
    when used with ALB(with targetgroups) does load balancing & autoscaling.
    so we are using this in the project
2. service - service, ServiceRegistry & Eureka