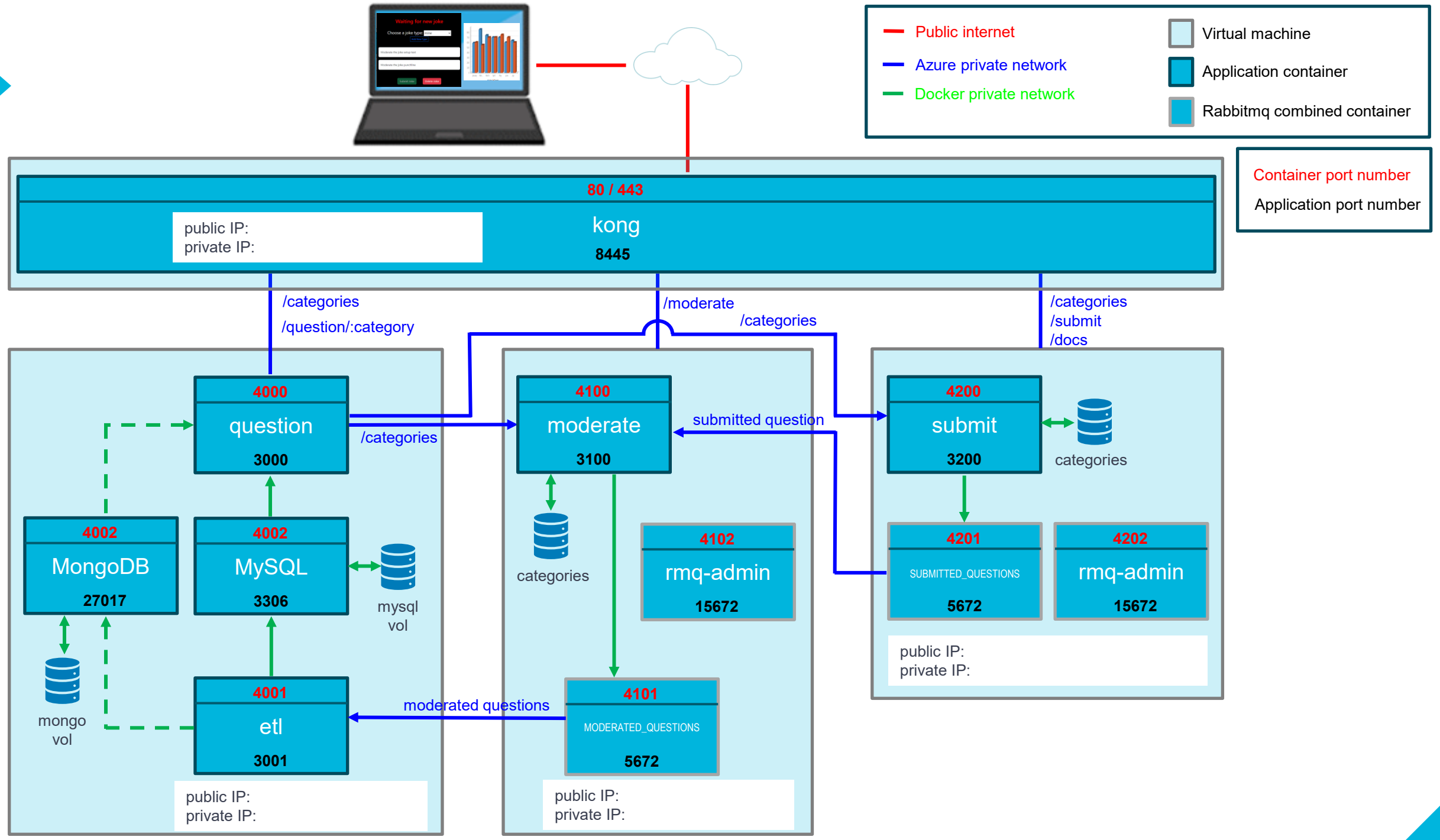
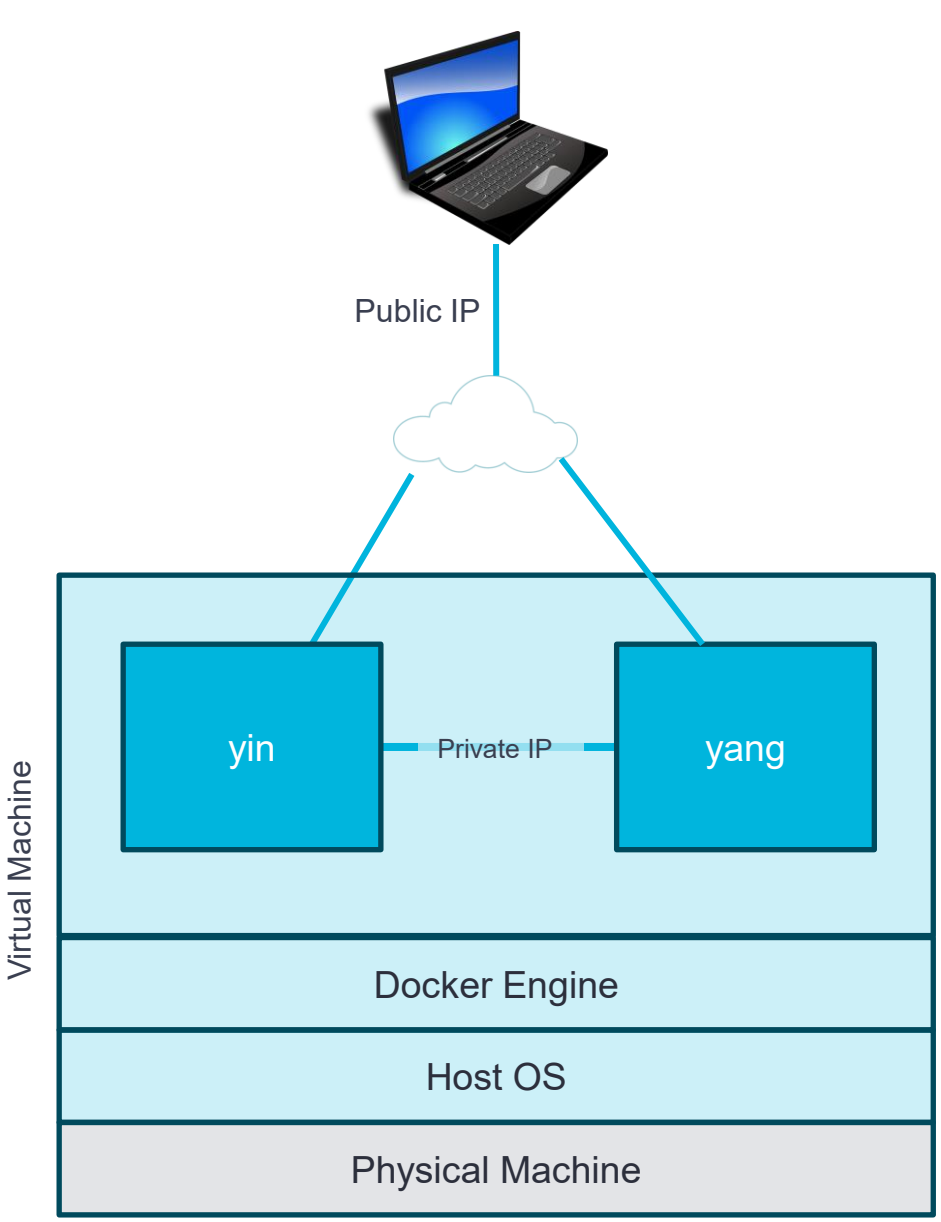
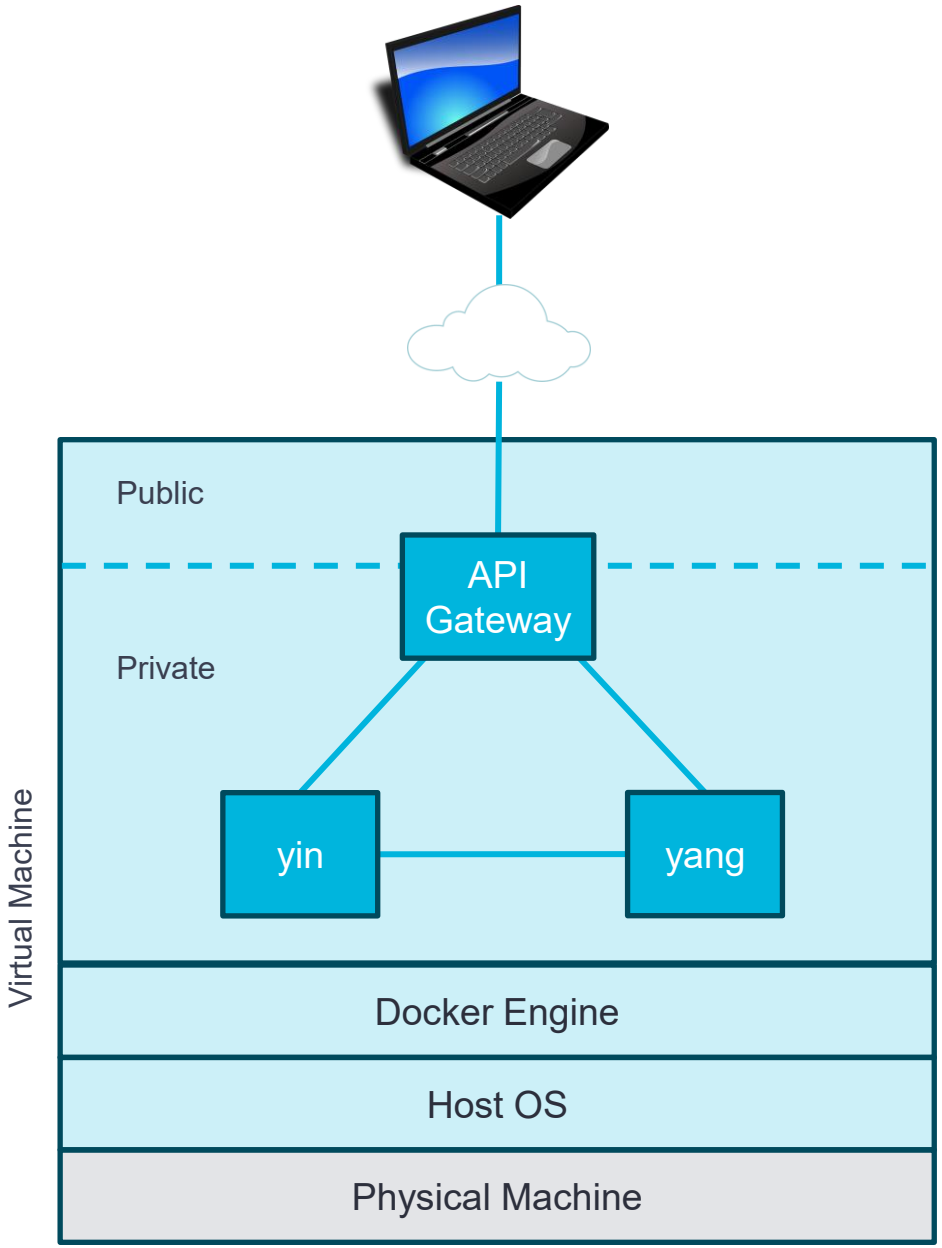
API Gateway

Learning objectives

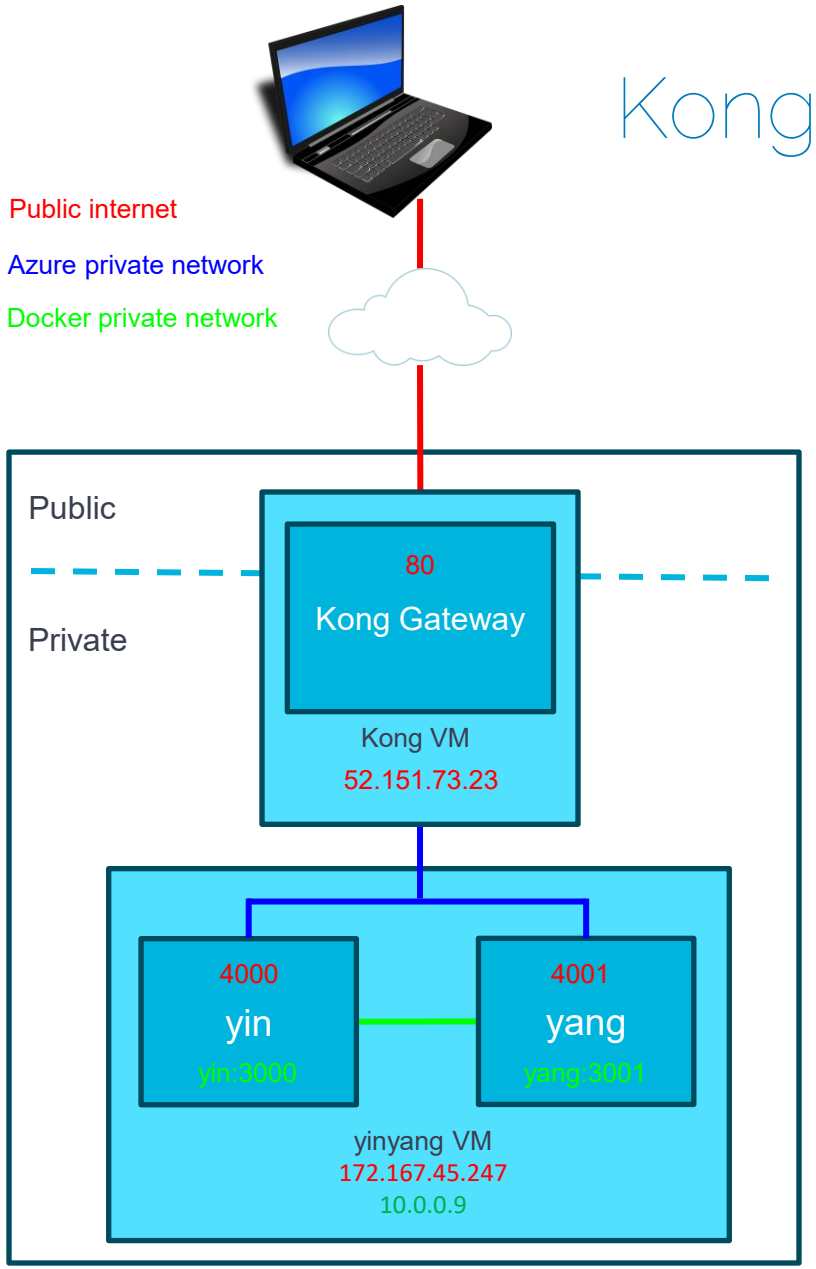
- Upon successfully completing this work, you should be able to:
  - Explain the benefits of using an API gateway including
    - Architectural
    - Security
    - Orchestration
    - Rate limiting
    - Monitoring
    - Authentication
    - Caching
  - Explain any disadvantages of using an API gateway
  - Implement an API gateway in local containers and in the cloud


Public internet
Azure private network
Docker private network
Virtual machine
Application container
Rabbitmq combined container
Container port number
Application port number
80 / 443
kong
8445
public IP:
private IP:
/categories
/question/:category
/moderate
/categories
/categories
/submit
/docs
4000
question
3000
/categories
4100
moderate
3100
submitted question
4200
submit
3200
categories
4002
MongoDB
27017
4002
MySQL
3306
mysql
vol
categories
4102
rmq-admin
15672
4201
SUBMITTED_QUESTIONS
5672
4202
rmq-admin
15672
public IP:
private IP:
mongo
vol
4001
etl
3001
moderated questions
4101
MODERATED_QUESTIONS
5672
public IP:
private IP:
public IP:
private IP:


Public IP
yin
Private IP
yang
Virtual Machine
Docker Engine
Host OS
Physical Machine

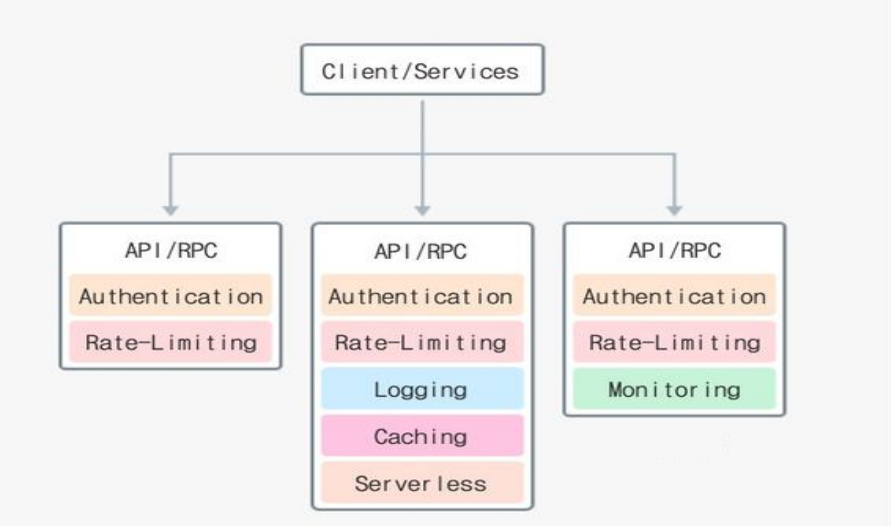

Public
API Gateway
Private
yin
yang
Virtual Machine
Docker Engine
Host OS
Physical Machine

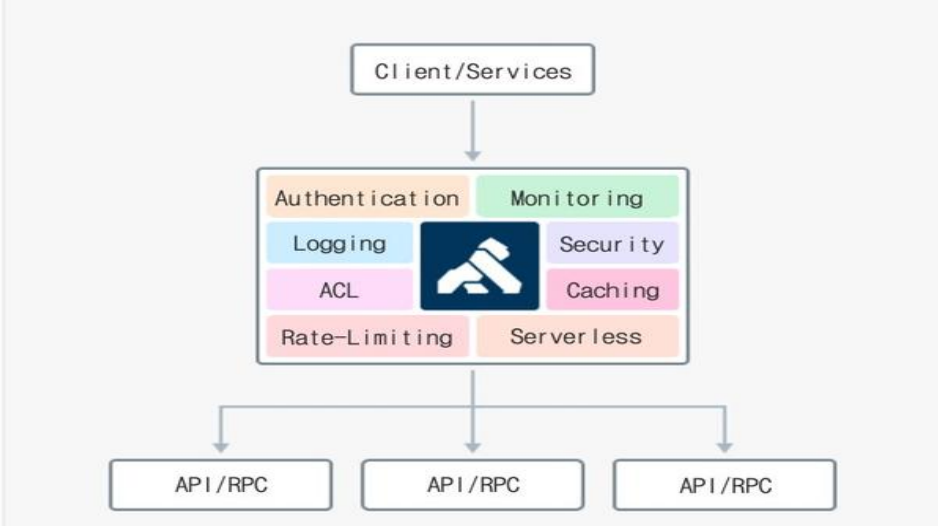
Kong API Gateway



- Single point of entry to all applications. No need for multiple ports and IPs
- Gateway acts as a reverse proxy, forwards requests to applications so manages what is available to the public
- All interaction between containers and gateway are private so don't need encryption unless you really need the extra security
- User connects to kong public ip with required path. e.g. **52.151.73.23:80/yin/yin.html** to get the main page and resources
- Kong will forward to the private ip of the yinyang vm. e.g. http://10.0.0.9:4000 on the Azure private network to the yin container
- The container forwards the request onto the node application at 3000
- The containers communicate between each other on a private docker network if on the same VM. e.g. yin makes a request to yang at its DNS name: yang:3001
- The gateway and applications are independently scalable

Kong high level features



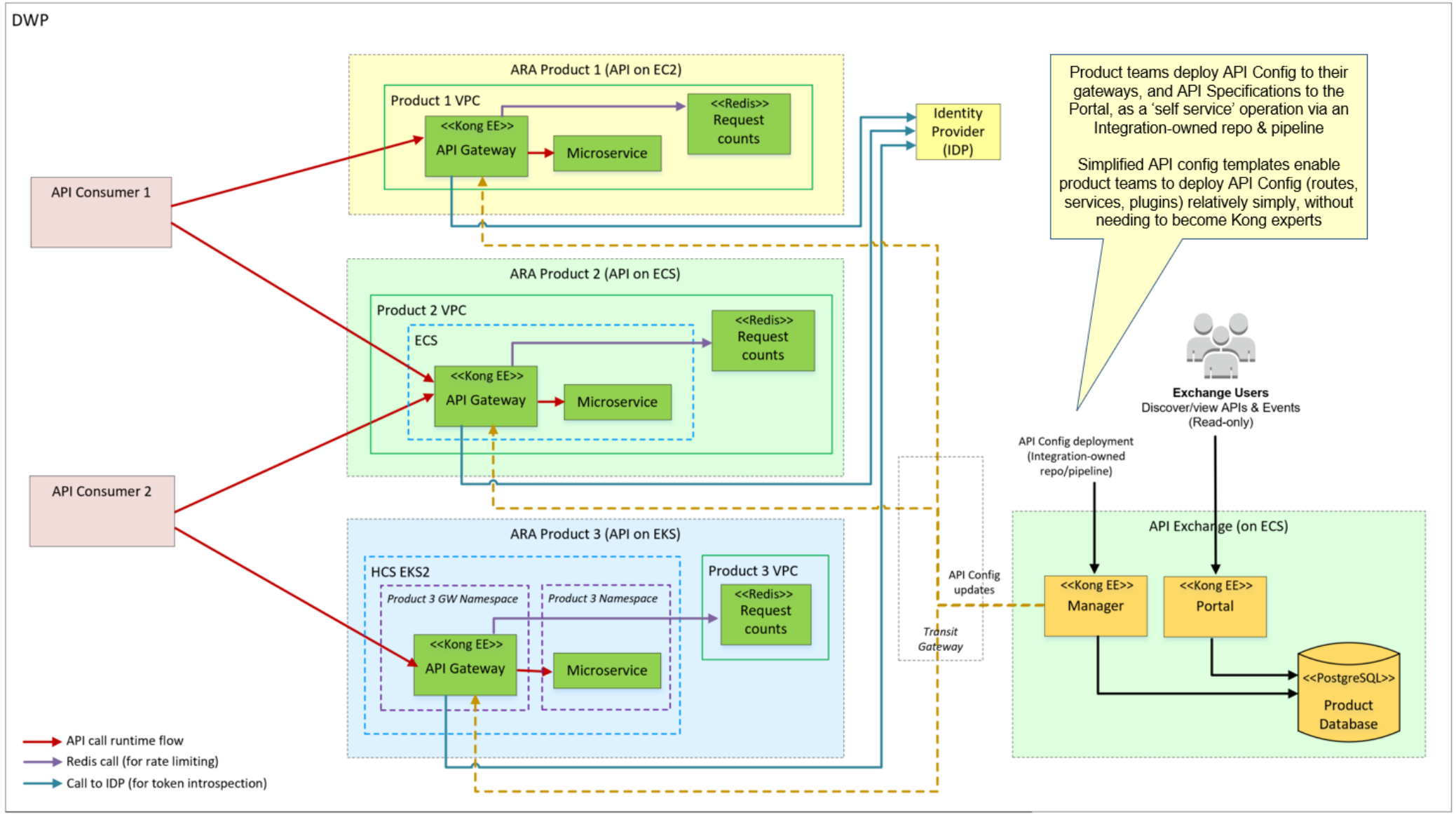


- Avoid replicating functionality such as rate limiting and authentication - add using Kong plugins
- Single entry point so https can be implemented once
- Aggregation of API calls - e.g. one request to multiple api calls, Kong can orchestrate that - reduces bandwidth
- Acts as a reverse proxy to protect services from the internet - users never see the endpoint IP
- Operates in two different modes: with database or without (db-less mode)
  - Database - typically Postgres or Cassandra - centrally stores all config info
  - In db-less, config is stored on each node - simpler deployment as no database dependency
  - db-less uses declarative code, db approach uses imperative via API

API Gateway properties

- Request Routing: route client requests to the appropriate microservices based on the URL or request parameters
- Load Balancing: load balancing evenly among multiple instances of the same microservice
- Security: authentication, authorization, and security in one place
- Rate Limiting: restricts the number of requests per min. Prevents abuse, protects backend resources
- Throttling: sets rate of processed requests, often by slowing down responses (queue) to smooth traffic spikes
- Caching: cache microservice responses. Reduce backend load and improve response times
- Logging and Analytics: provide logging to analyse traffic patterns and API usage
- Aggregation: aggregate multiple microservice responses into a single client response
  - reduces latency for client as they can make one rather than several individual calls

- Disadvantages - not many but they need to be considered
  - Single point of failure. Must be designed to be resilient and scalable - multi-node with load balancer
  - Extra hop may introduce a bit of extra latency
  - Adds complexity to the architecture - needs configuring, monitoring and maintenance (e.g. patching)
  - Extra cost: cost to run and knowledgeable staff to understand the technology so adds to cost
  - Security, this is an advantage but, as all services are accessible from here, it increases the attack surface for hackers so security of the gateway must be carefully designed


DWP
ARA Product 1 (API on EC2)
Product 1 VPC
<<Kong EE>>
API Gateway
Microservice
<<Redis>>
Request counts
Identity Provider (IDP)
Product teams deploy API Config to their gateways, and API Specifications to the Portal, as a 'self service' operation via an Integration-owned repo & pipeline
Simplified API config templates enable product teams to deploy API Config (routes, services, plugins) relatively simply, without needing to become Kong experts
API Consumer 1
ARA Product 2 (API on ECS)
Product 2 VPC
ECS
<<Kong EE>>
API Gateway
Microservice
<<Redis>>
Request counts
Exchange Users
Discover/view APIs & Events (Read-only)
API Config deployment (Integration-owned repo/pipeline)
API Consumer 2
ARA Product 3 (API on EKS)
HCS EKS2
Product 3 GW Namespace
Product 3 Namespace
Product 3 VPC
<<Kong EE>>
API Gateway
Microservice
<<Redis>>
Request counts
API Config updates
Transit Gateway
API Exchange (on ECS)
<<Kong EE>>
Manager
<<Kong EE>>
Portal
<<PostgreSQL>>
Product Database
API call runtime flow
Redis call (for rate limiting)
Call to IDP (for token introspection)

- The gateway is relatively simple to use:

  routes are defined in kong.yaml and associated with a forwarding address

```yaml
- name: yin
  url: http://10.0.2.4:4000
  routes:
    - name: yin
      paths:
        - /yin
      strip_path: false
```

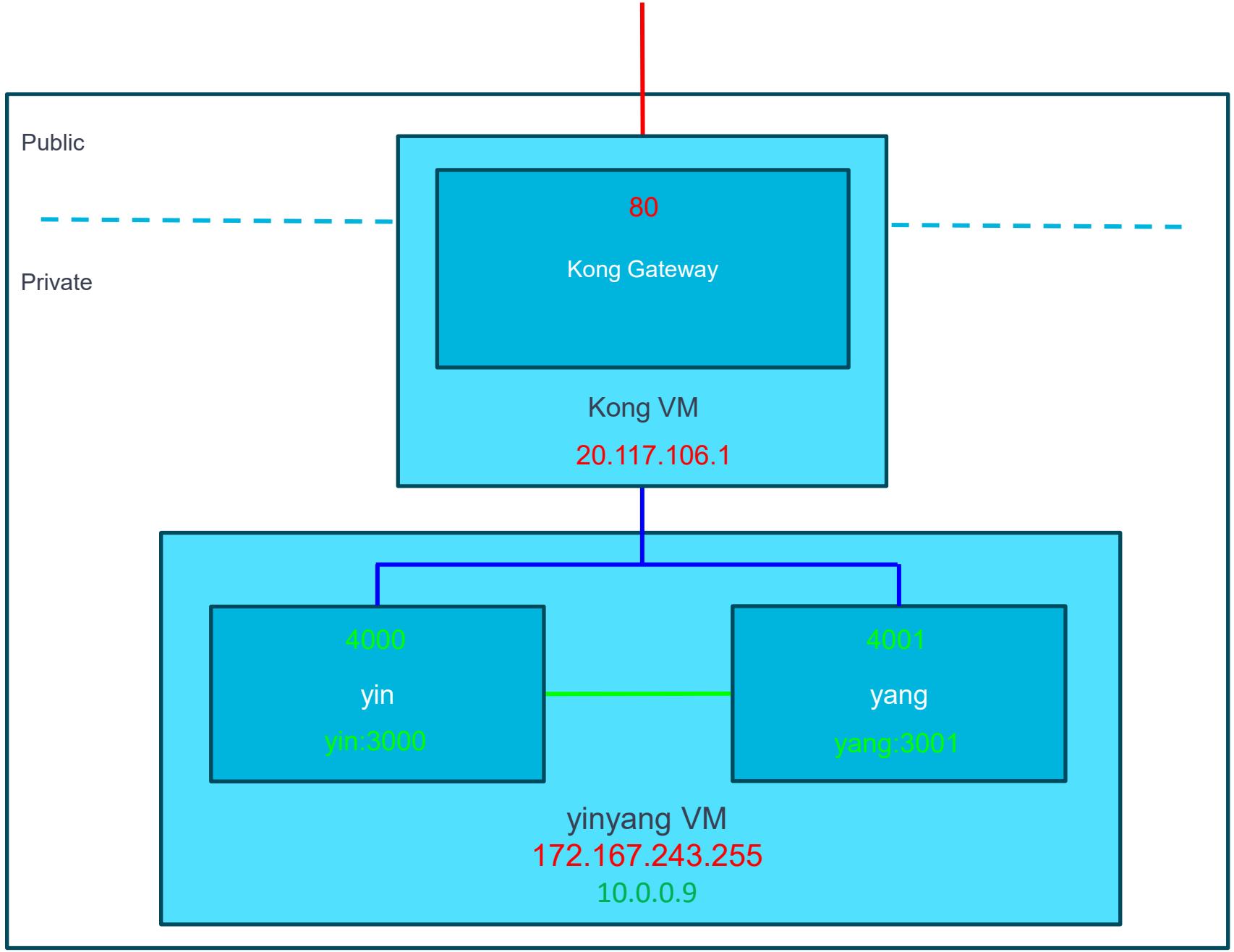
```yaml
- name: yang
  url: http://10.0.2.4:4001
  routes:
    - name: yang
      paths:
        - /yang
      strip_path: true
```

  - http:// 52.151.73.23/yin is matched on path then origin changed to http://10.0.2.4:4000/yin because strip_path is false
  - http://52.151.73.23/yang/home is forwarded to http://10.0.2.4:4001/home because strip_path is true so yang is removed
    - strip_path is **true by default** and will remove from the source path, the path element specified in the path property
    - Can be useful for example, for versioning:
      kongIP/api/v1-1-2/books/:media could be routed to 10.0.2.3/books/:media by specifying /api/v1-1-2 in the paths and strip_path: true
    - kongIP/api/v1-2-0/books/:media could be routed to 10.0.3.5/books/:media as kong will match the source path to the paths definition to find a match, then remove the source path and replace the IP address
  - So, it's up to you what you use - strip_path enables flexibility in how traffic is forwarded
  - For more complex path filtering, regex can be used

Public internet

Azure private network

Docker private network



/demos/architecture/API gateway kong