Cloud Agnostic Multi-Tenant SaaS Applications - Challenges & Solutions

Abhay Dutt Paroha

Software Team Leader, SLB

Conf42

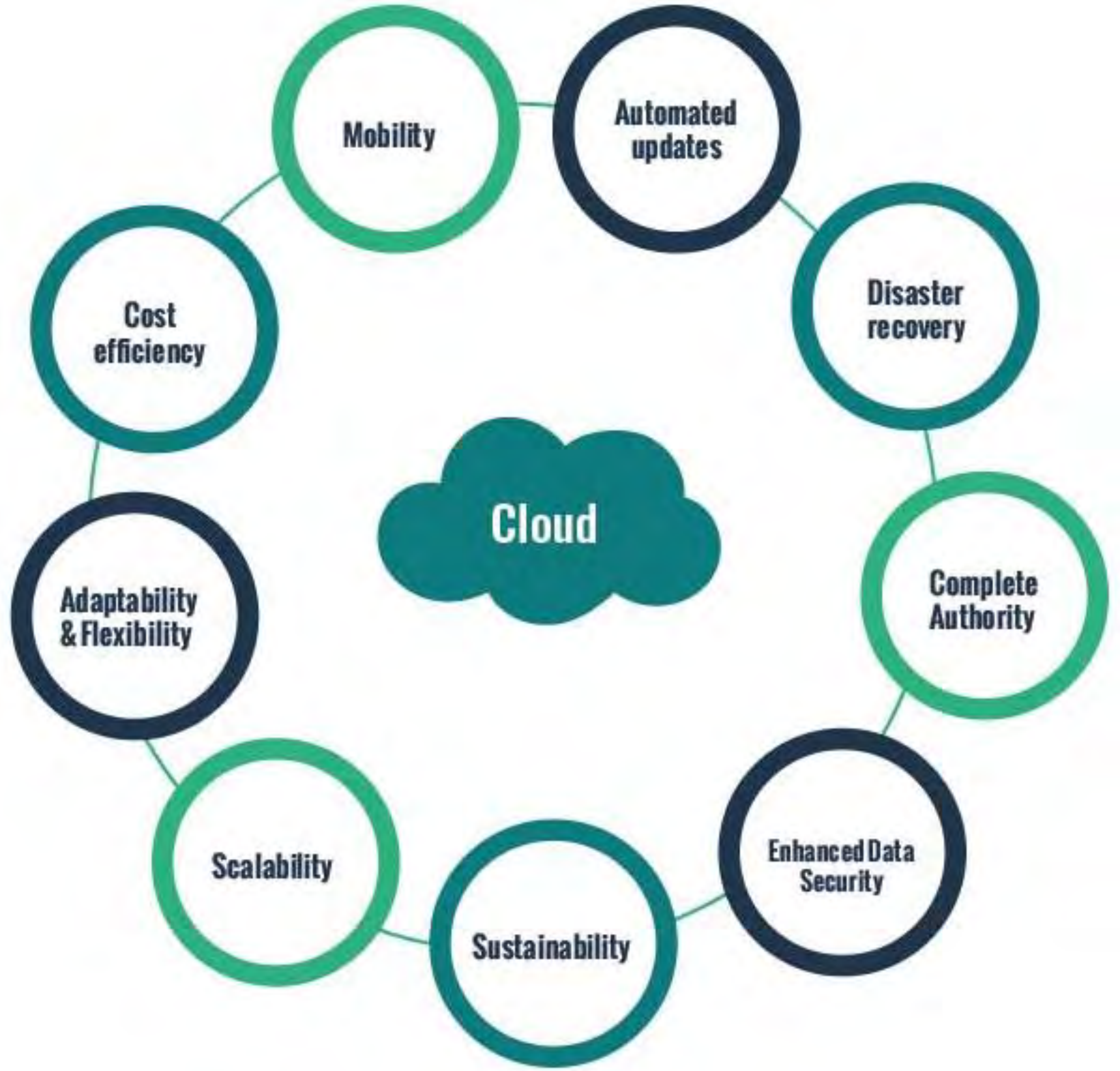
A word about me

- 3+ years managing engineering teams working on real-time upstream operational data ingestion and delivery
- 14 years of experience in various aspects of the oilfield (operational and technical)

*Disclaimer: This session is not about cloud infrastructure; it is about using cloud to build applications.

What is Cloud Agnostic?

Applications, Services, Systems, Tools, and Workloads that are designed to be compatible with multiple cloud providers, rather than cloud-native being tied to a specific cloud platform

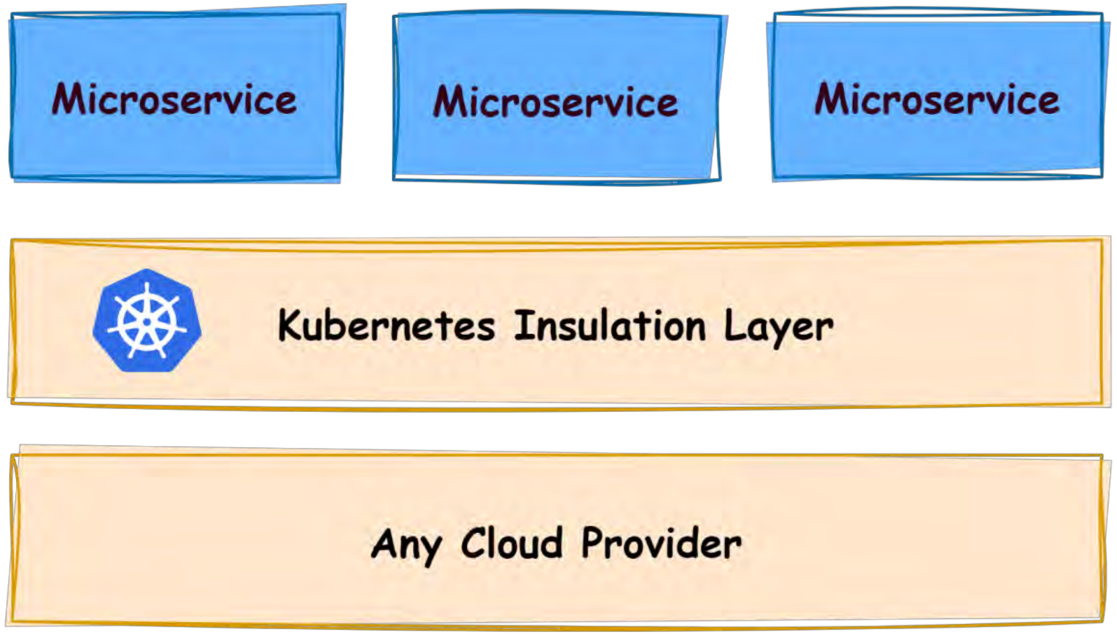


Why *Cloud*?

By 2028, cloud computing will shift from being a technology disruptor to becoming a necessary component for maintaining business competitiveness - *Gartner*

Cloud agnostic pros and cons

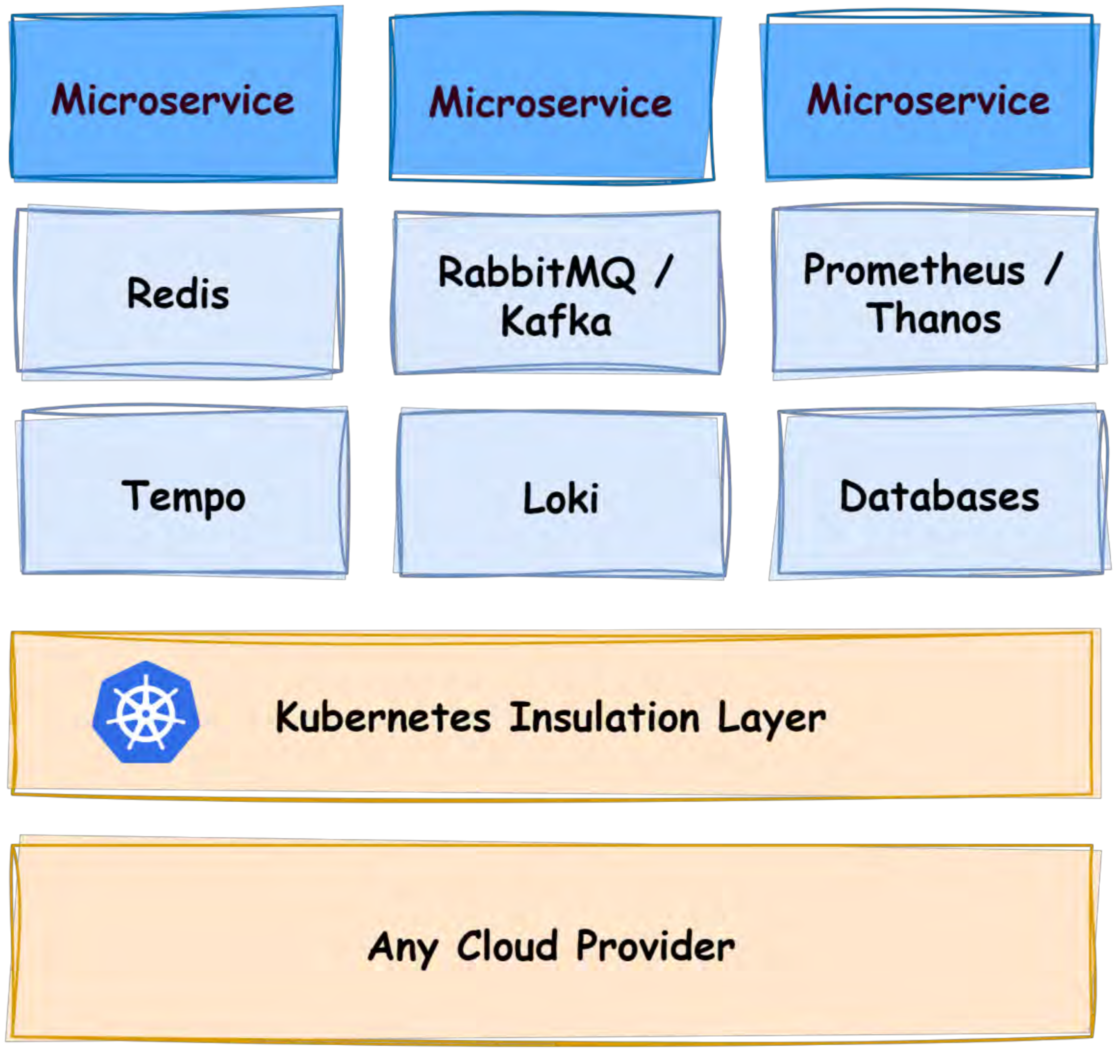
- Pros
  - Avoid the risk of vendor lock-in
  - Performance
  - Flexibility
  - Resilience
- Cons
  - Challenging implementations
  - Time to market



How to design cloud agnostic architecture?

Use Kubernetes and you are done!

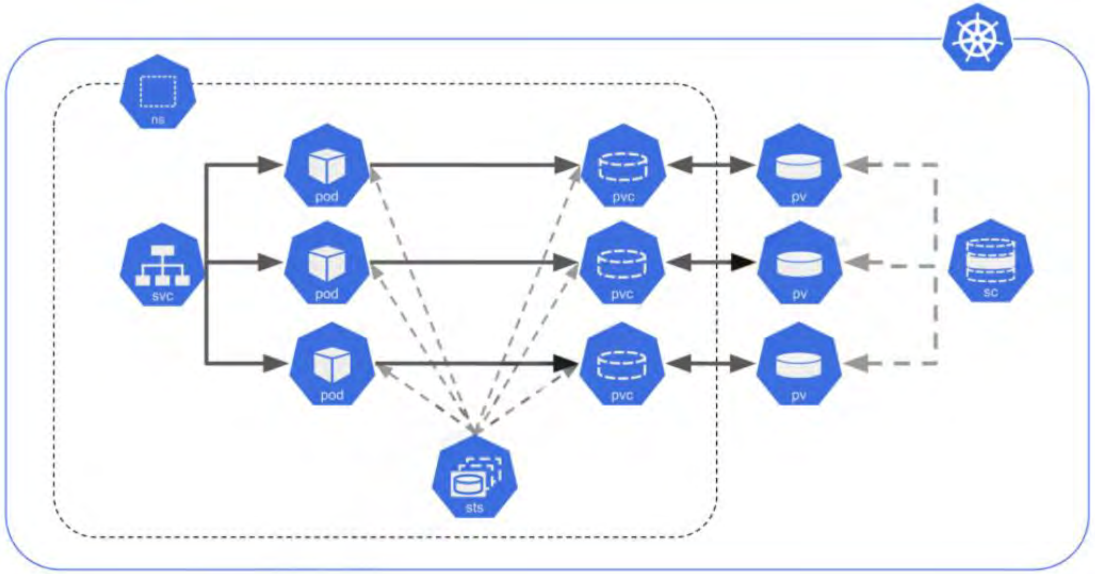
- Build services as containerized workloads, our friend Docker.
- Deploy containers to the Kubernetes product offered by the cloud vendor. (AKS, EKS, GKE)
- New capability -> New Container
- Switch providers as long as Kubernetes is available



New capability -> New Container

- Need Messaging? RabbitMQ or Kafka in containers
- Need Cache? Redis High Availability cluster in containers
- Need a RDBMS? PostgreSQL in containers
- Need an Object Storage like AWS S3? Minio in containers
- Need Monitoring? ELK (Elasticsearch, Logstash, Kibana) stack in containers

Run a stateful application on Kubernetes



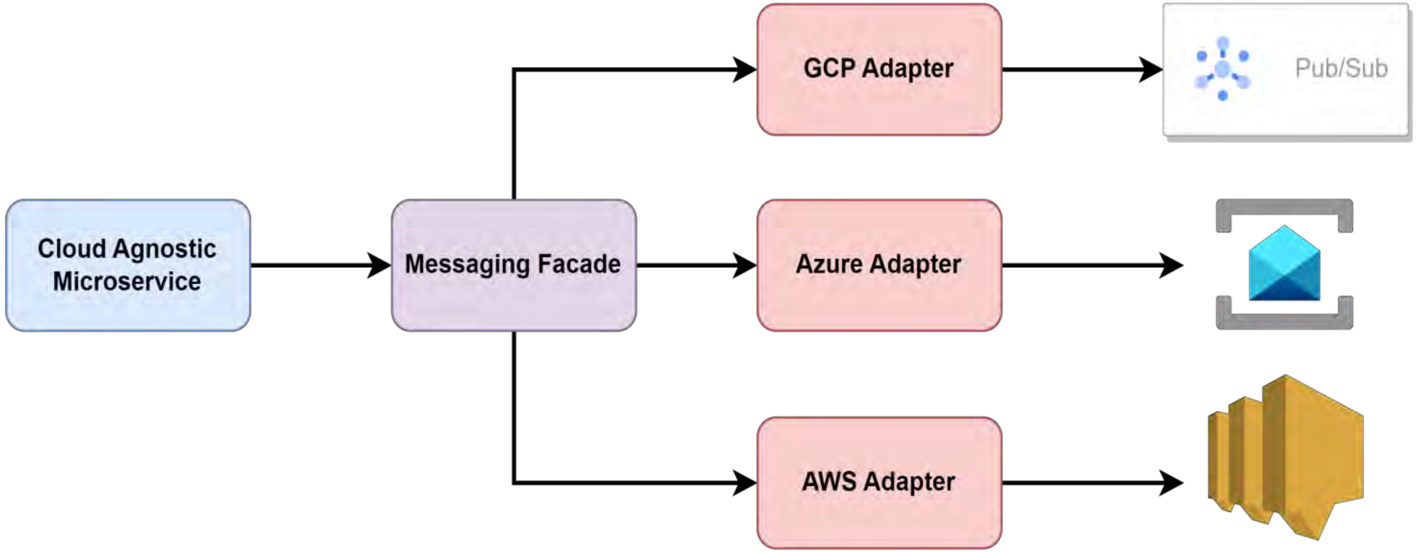
- StatefulSets
  - Startup, Scale-up/down, Rolling Upgrades, Termination - Ordered operations with ordinal index
  - Unique network ID/name to maintain affinity
  - Persistent storage disk linked to the ordinal index – network, local, cloud
  - Headless service (don’t forget network policies for security)

Other challenges

- Cloud capabilities
  - Different cloud providers -> different data centers, data residency, GDPR
  - Failover, resilience, and latency -> depend on the location of data centers
- Networking

  Unlike AWS and Azure, GCP's Virtual Private Cloud resources are not tied to any specific region. It’s a global resource.
- Data egress cost -> Ingress is free, but egress can be expensive
- Infrastructure as a code

Loosely coupled architecture



Cloud Agnostic Microservice

Messaging Facade

GCP Adapter

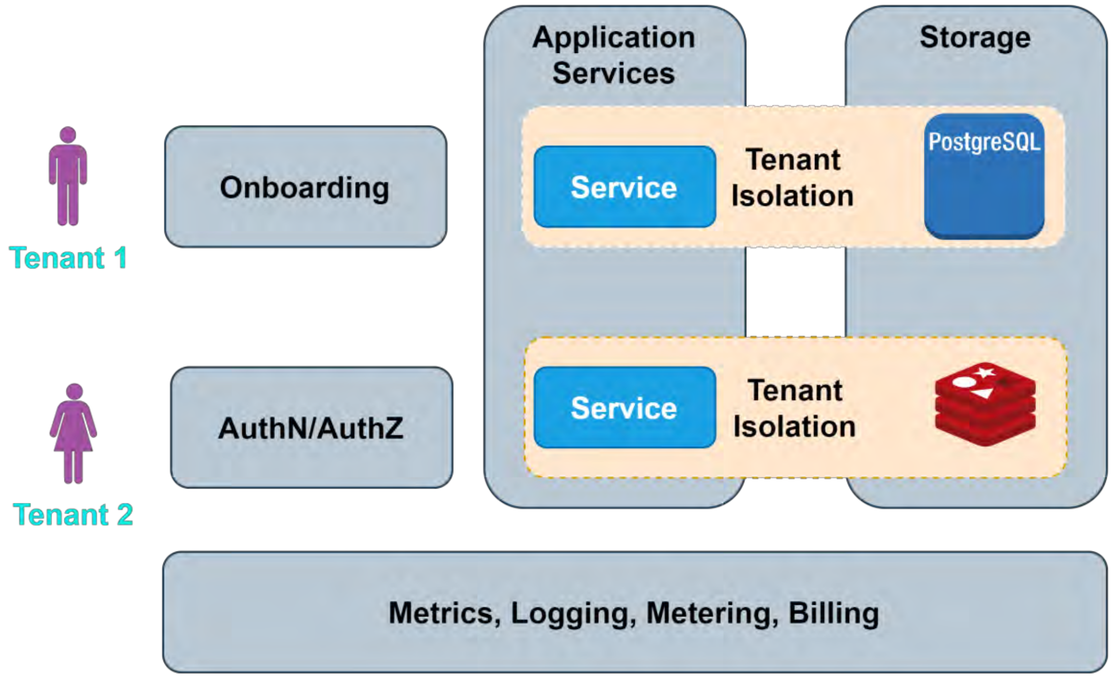
Azure Adapter

AWS Adapter

A facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes.

Strategic lock-in

- Identify the areas where lock-in must be kept to a minimum.
- Only use products, that have corresponding counterparts on the other platforms.
    - Databases
        - SQL DB -> GCP Cloud SQL, Azure database for PostgreSQL
        - Runtime -> GCP GKE, AWS EKS
        - Serverless -> Knative
        - Timeseries database -> GCP BigTable, AWS DynamoDB



Tenant aware operations dashboard

Fundamentals of SaaS

- Onboarding
- AuthN/AuthZ
- Data partitioning
- Tenant isolation
- Metering and billing
- Tenant aware operation models

Multi-tenant impact

Frontend

Authentication

Routing

Feature Flags

API Gateway

Authorization

Throttling

Caching

Business logic

Metrics

Logging

Metering

Data persistence

Data Access, Partitioning and Isolation

Backup/restore

Infrastructure

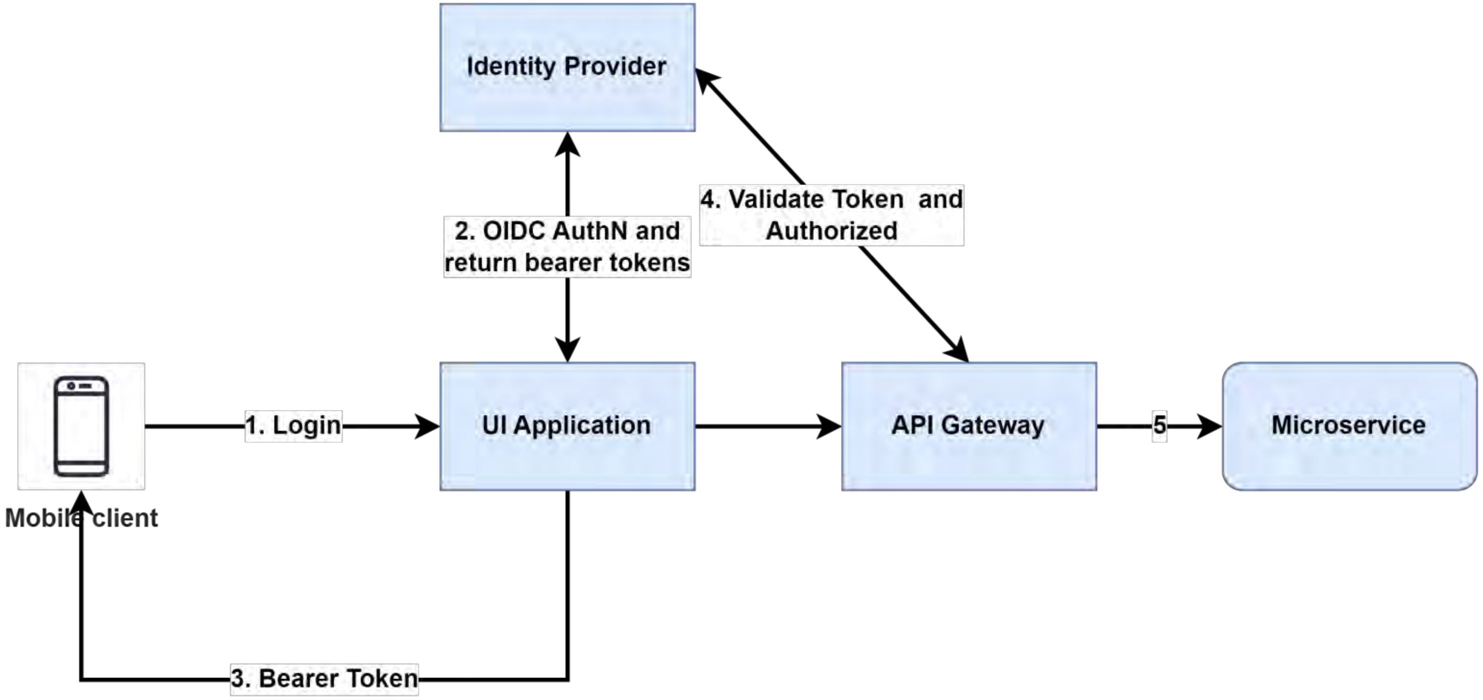
Provisioning

Isolation

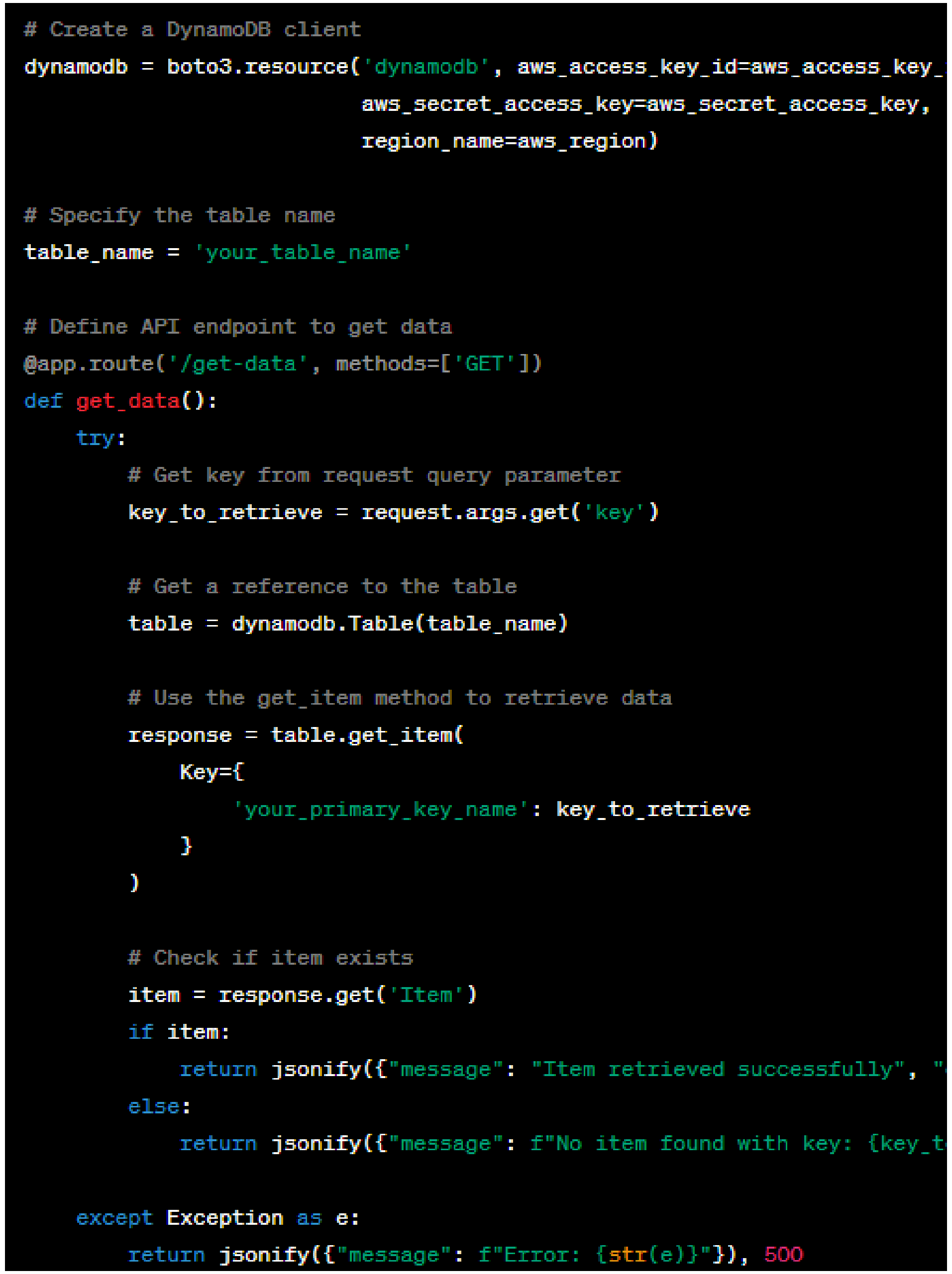
Maintenance

Tenant lifecycle

A normal microservice flow

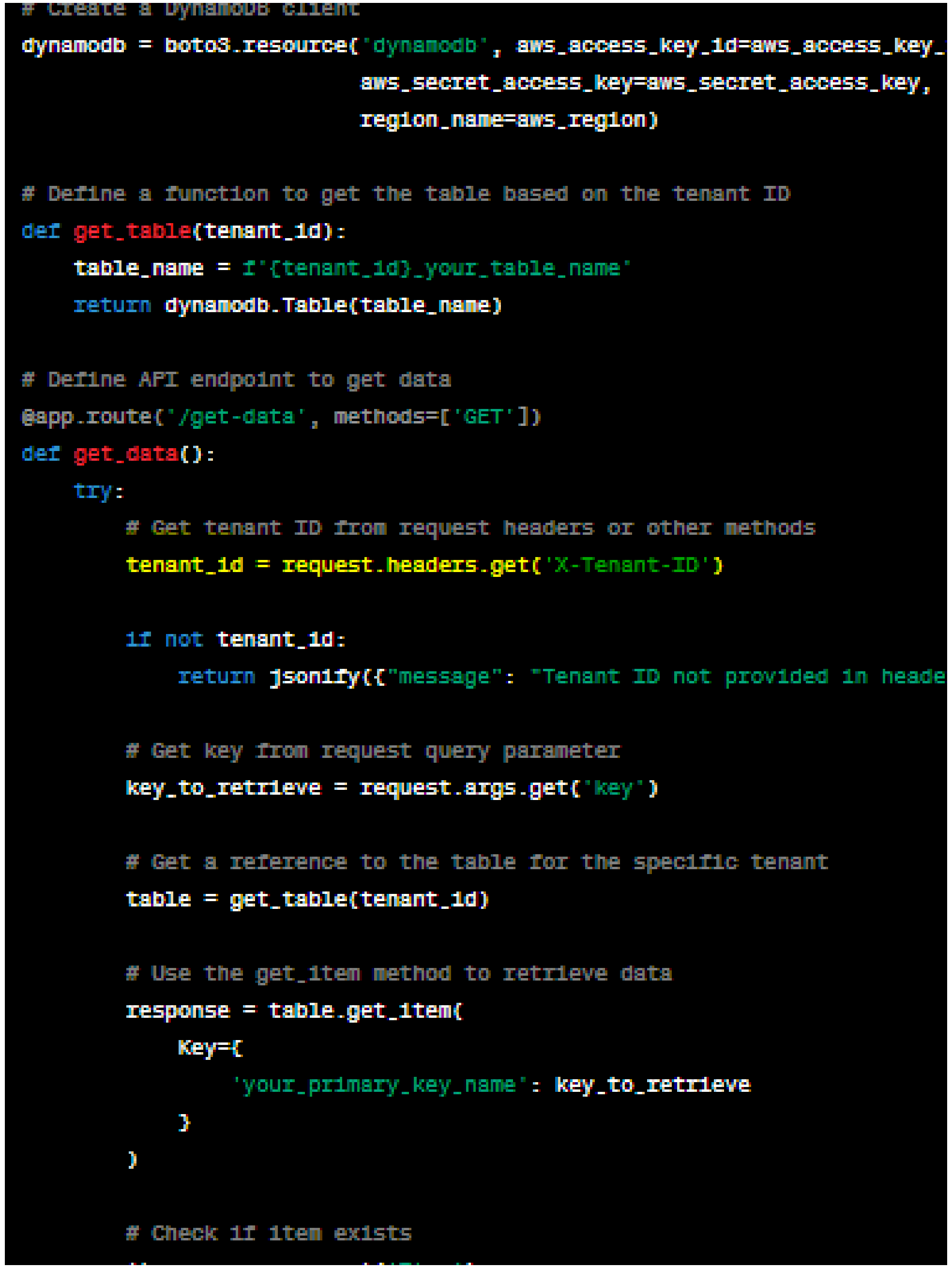


Non-SaaS microservice



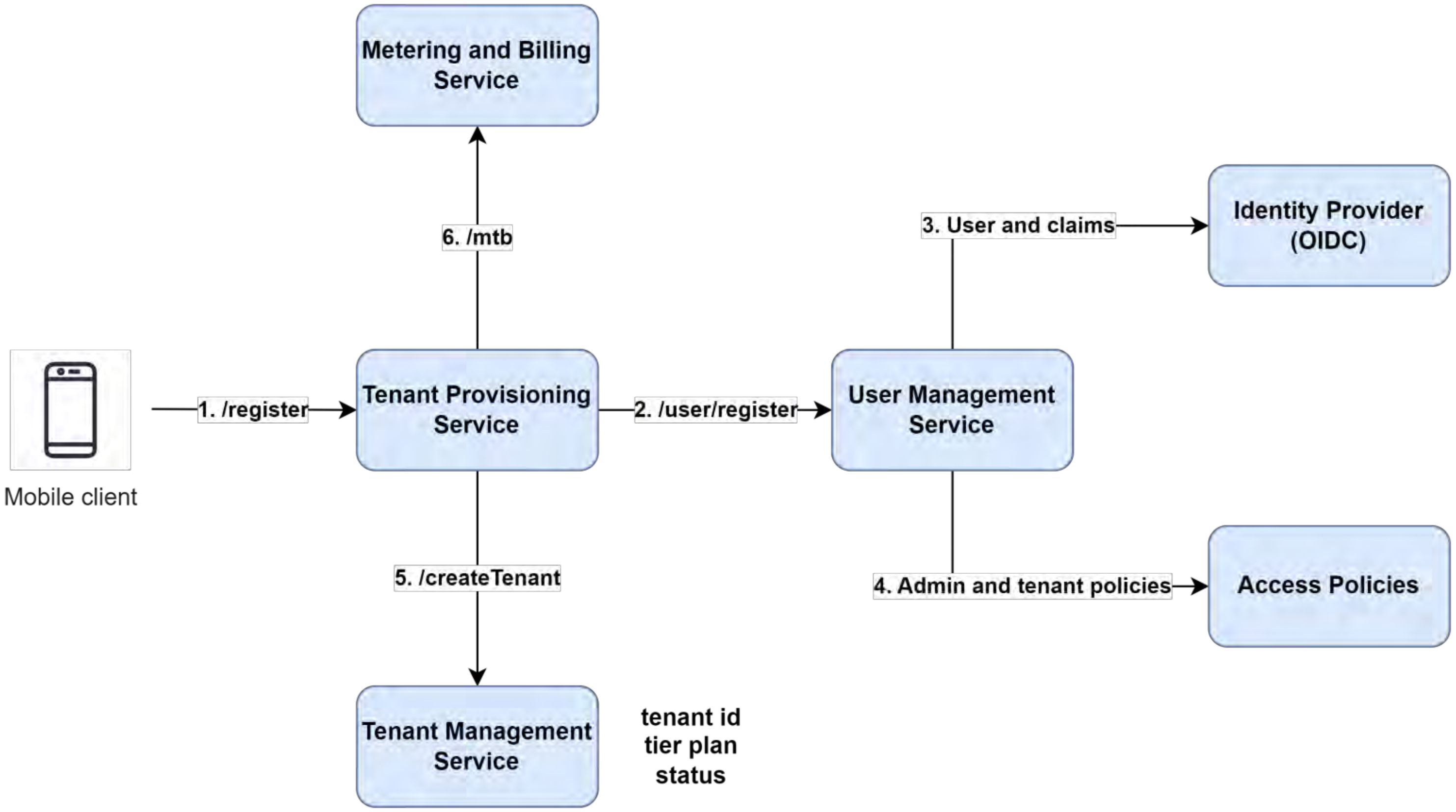
```
# Create a DynamoDB client
dynamodb = boto3.resource('dynamodb', aws_access_key_id=aws_access_key_
                          aws_secret_access_key=aws_secret_access_key,
                          region_name=aws_region)

# Specify the table name
table_name = 'your_table_name'

# Define API endpoint to get data
@app.route('/get-data', methods=['GET'])
def get_data():
    try:
        # Get key from request query parameter
        key_to_retrieve = request.args.get('key')

        # Get a reference to the table
        table = dynamodb.Table(table_name)

        # Use the get_item method to retrieve data
        response = table.get_item(
            Key={
                'your_primary_key_name': key_to_retrieve
            }
        )

        # Check if item exists
        item = response.get('Item')
        if item:
            return jsonify({"message": "Item retrieved successfully", "
        else:
            return jsonify({"message": f"No item found with key: {key_t

    except Exception as e:
        return jsonify({"message": f"Error: {str(e)}"}), 500
```

SaaS microservice



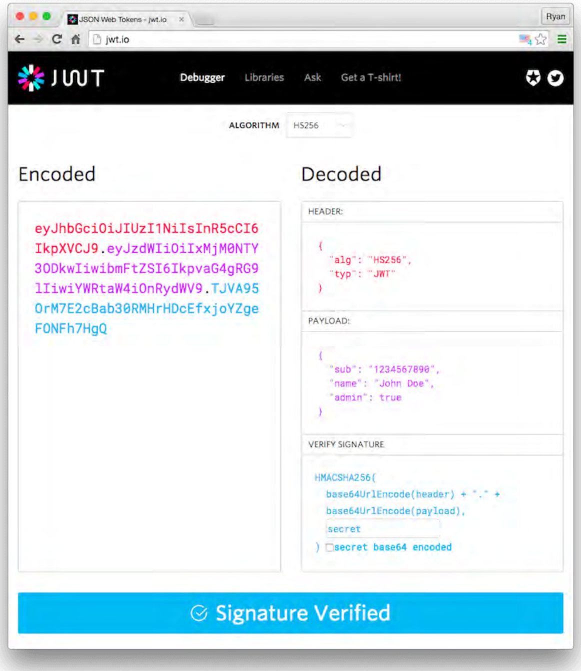
```
# Create a DynamoDB client
dynamodb = boto3.resource('dynamodb', aws_access_key_id=aws_access_key_
                          aws_secret_access_key=aws_secret_access_key,
                          region_name=aws_region)

# Define a function to get the table based on the tenant ID
def get_table(tenant_id):
    table_name = f'{tenant_id}_your_table_name'
    return dynamodb.Table(table_name)

# Define API endpoint to get data
@app.route('/get-data', methods=['GET'])
def get_data():
    try:
        # Get tenant ID from request headers or other methods
        tenant_id = request.headers.get('X-Tenant-ID')

        if not tenant_id:
            return jsonify({"message": "Tenant ID not provided in heade

        # Get key from request query parameter
        key_to_retrieve = request.args.get('key')

        # Get a reference to the table for the specific tenant
        table = get_table(tenant_id)

        # Use the get_item method to retrieve data
        response = table.get_item(
            Key={
                'your_primary_key_name': key_to_retrieve
            }
        )

        # Check if item exists
```

Provision a tenant



Acquiring Tenant Context

- JWT token (JSON web token)
- In the URL as a query string parameter
- Request Header
- Separate microservice



ALGORITHM HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```

Decoded

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

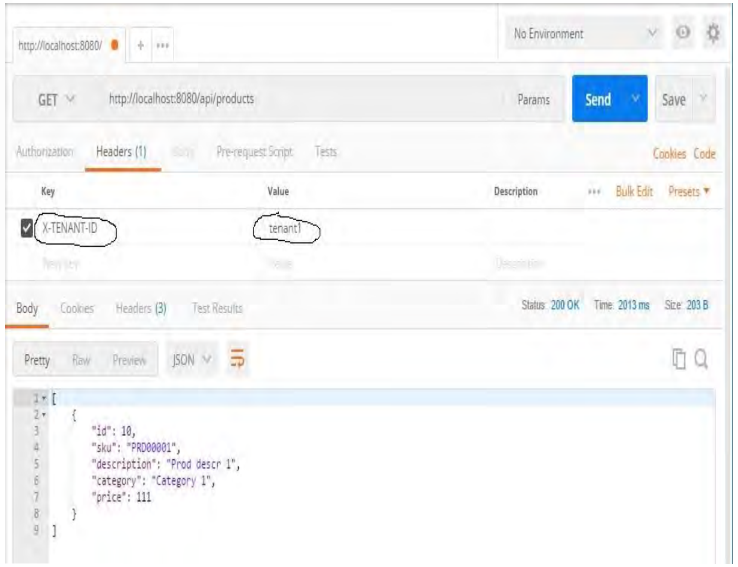
PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) □secret base64 encoded
```

Signature Verified

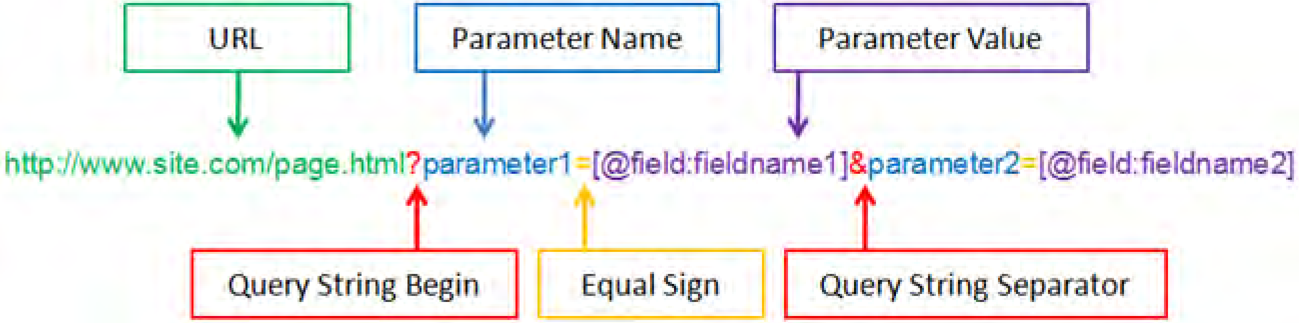


GET http://localhost:8080/api/products

Key	Value	Description
☑ X-TENANT-ID	tenant1	

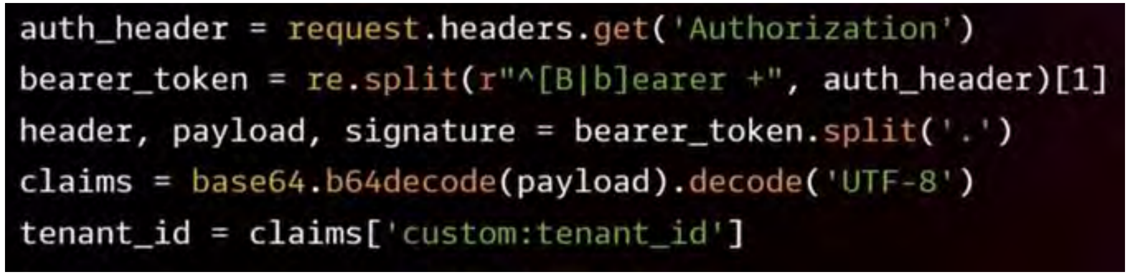
Status: 200 OK Time: 2013 ms Size: 203 B

```
[
    {
        "id": 10,
        "sku": "PRD00001",
        "description": "Prod descr 1",
        "category": "Category 1",
        "price": 111
    }
]
```

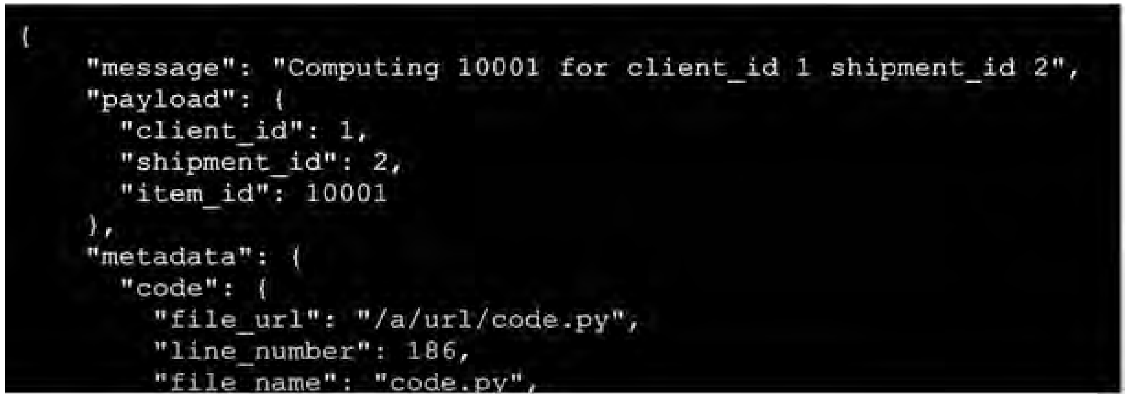

URL
Parameter Name
Parameter Value
http://www.site.com/page.html?parameter1=[@field:fieldname1]&parameter2=[@field:fieldname2]
Query String Begin
Equal Sign
Query String Separator

Common libraries

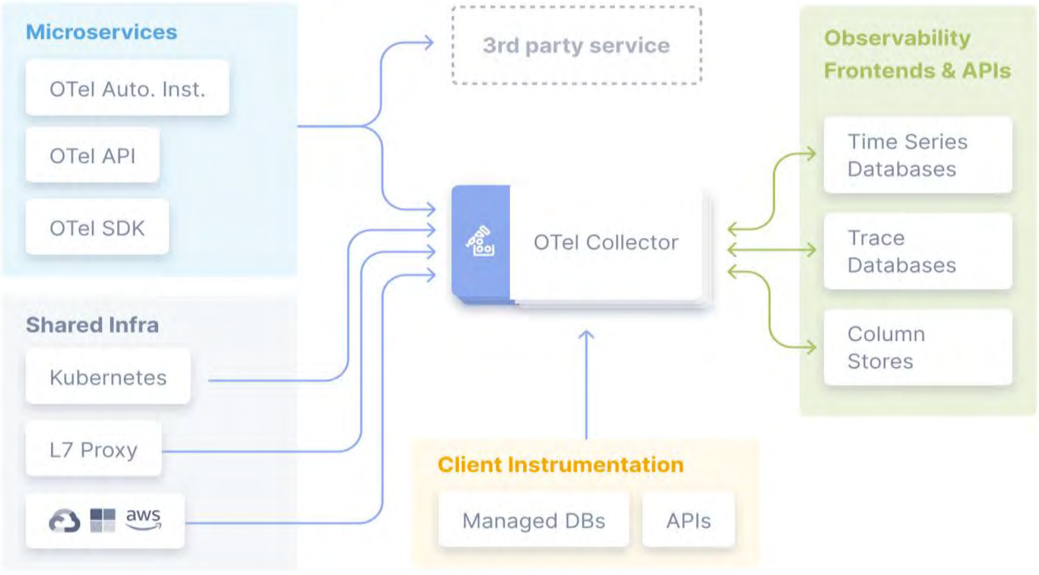
- Get tenant ID from JWT token
- Structured logging
- Metrics



```
auth_header = request.headers.get('Authorization')
bearer_token = re.split(r"^[B|b]earer +", auth_header)[1]
header, payload, signature = bearer_token.split('.')
claims = base64.b64decode(payload).decode('UTF-8')
tenant_id = claims['custom:tenant_id']
```

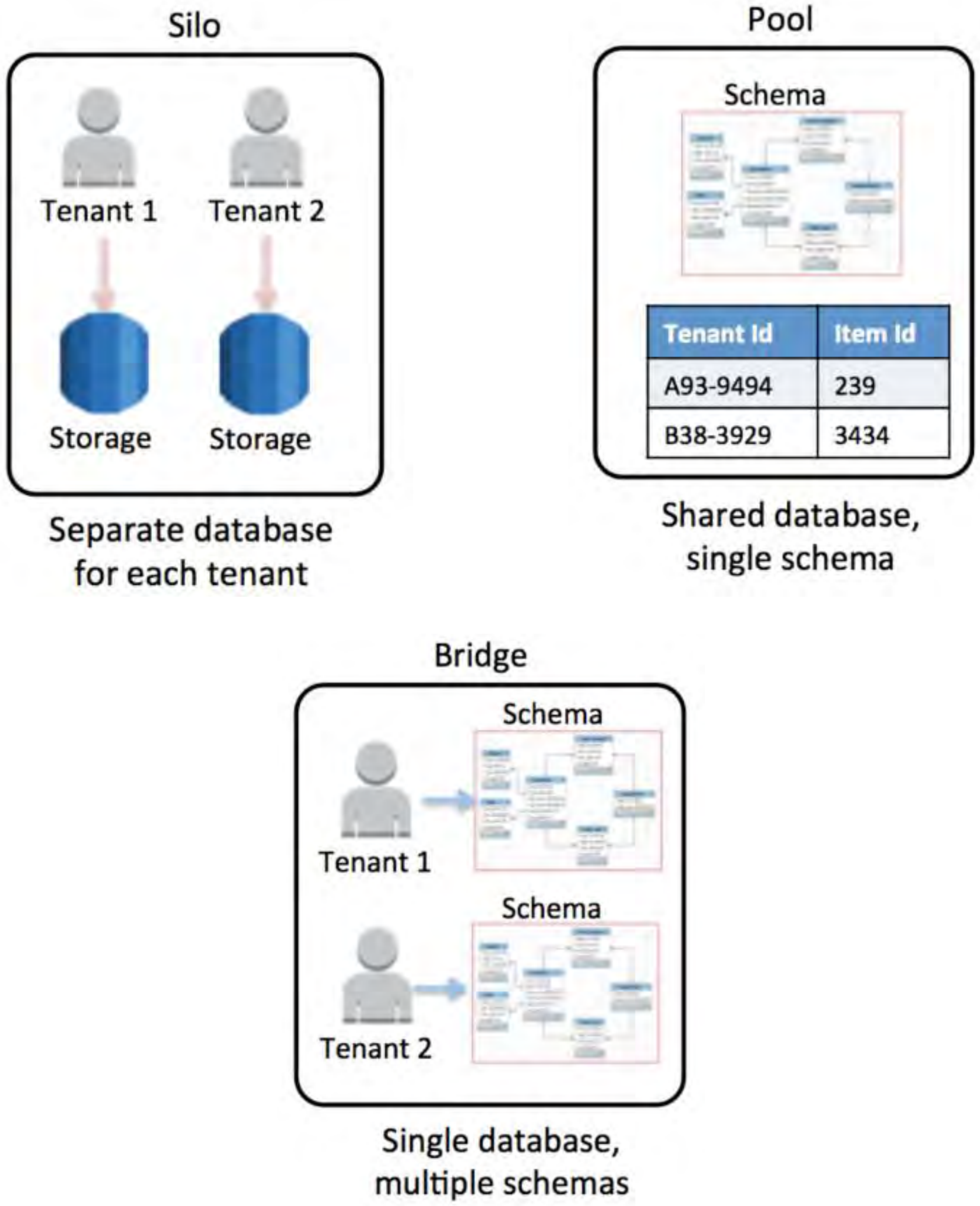


```
{
    "message": "Computing 10001 for client_id 1 shipment_id 2",
    "payload": {
      "client_id": 1,
      "shipment_id": 2,
      "item_id": 10001
    },
    "metadata": {
      "code": {
        "file_url": "/a/url/code.py",
        "line_number": 186,
        "file_name": "code.py",
```



Data partitioning

- A microservice-based decision
  - Compliance & security
  - Performance
  - Data distribution
  - Noisy neighbor



Tenant Id	Item Id
A93-9494	239
B38-3929	3434

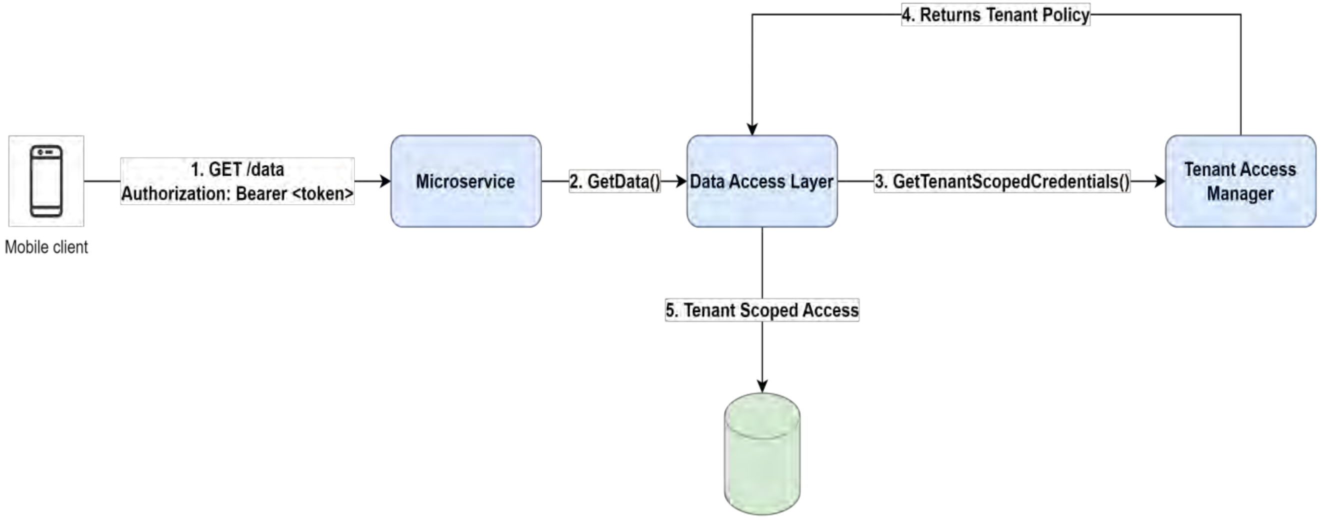
Separate database for each tenant

Shared database, single schema

Single database, multiple schemas

Tenant Isolation

- Silo Model – every tenant gets their own environment
- Pool Model – isolation through runtime policies
  - Before touching any resource, get the policies assigned to a specific tenant



Take aways

- Strategic lock-in
- Loosely coupled architecture
- Tenant lifecycle
- Tenant context
- Data partition
- Tenant isolation

Thank You!