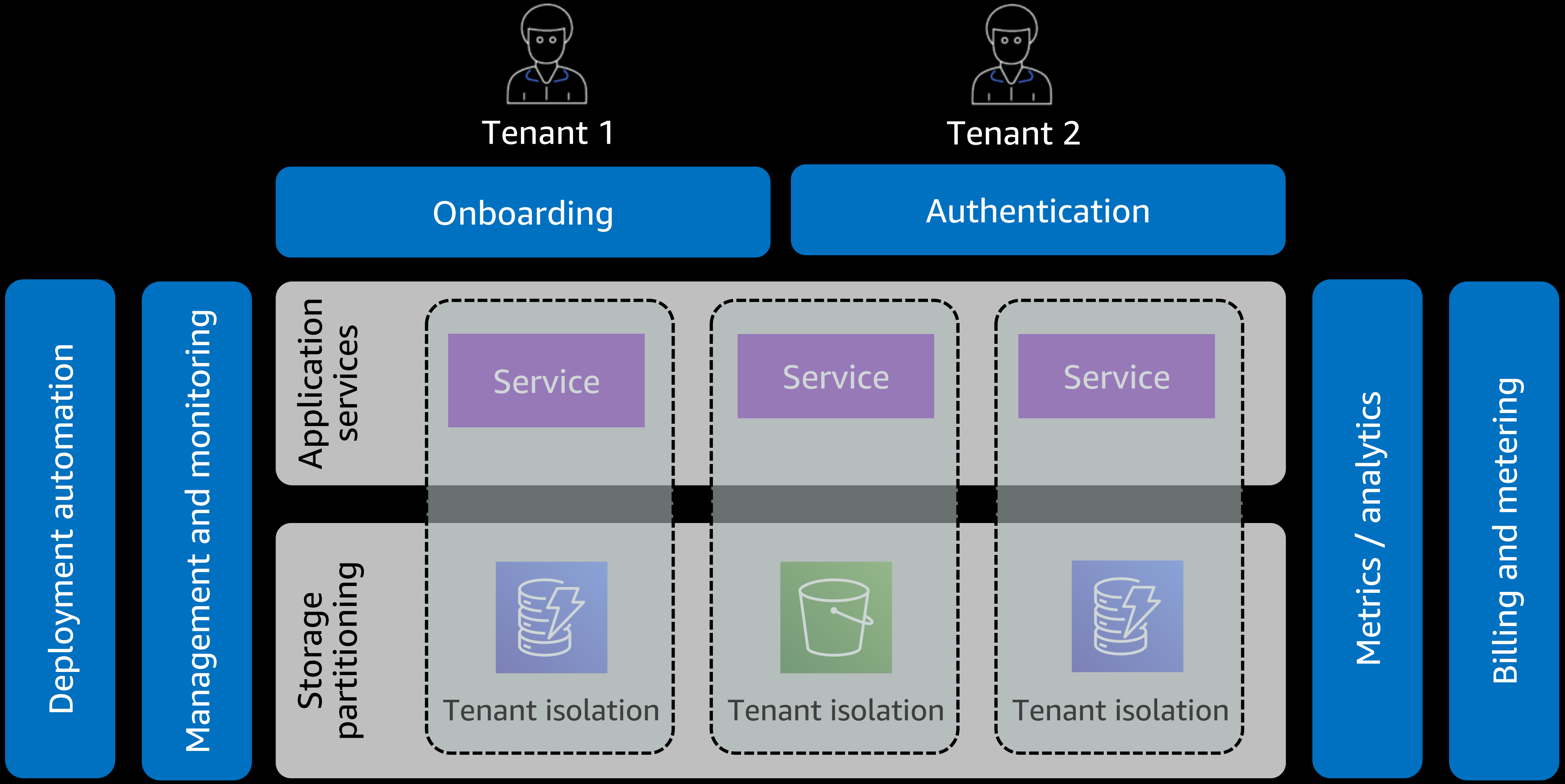
ARC314

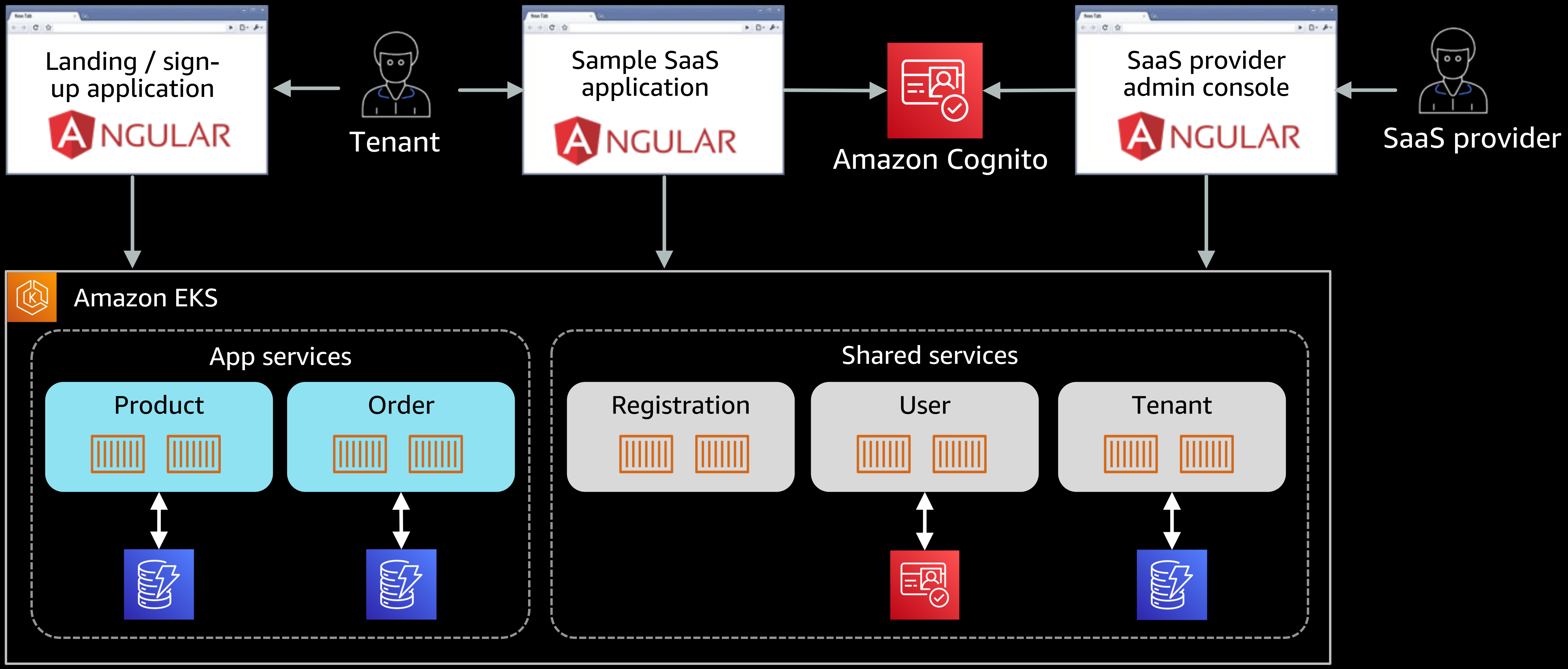
Inside Amazon EKS SaaS: Building multi-tenant solutions with EKS

AWS re:Invent

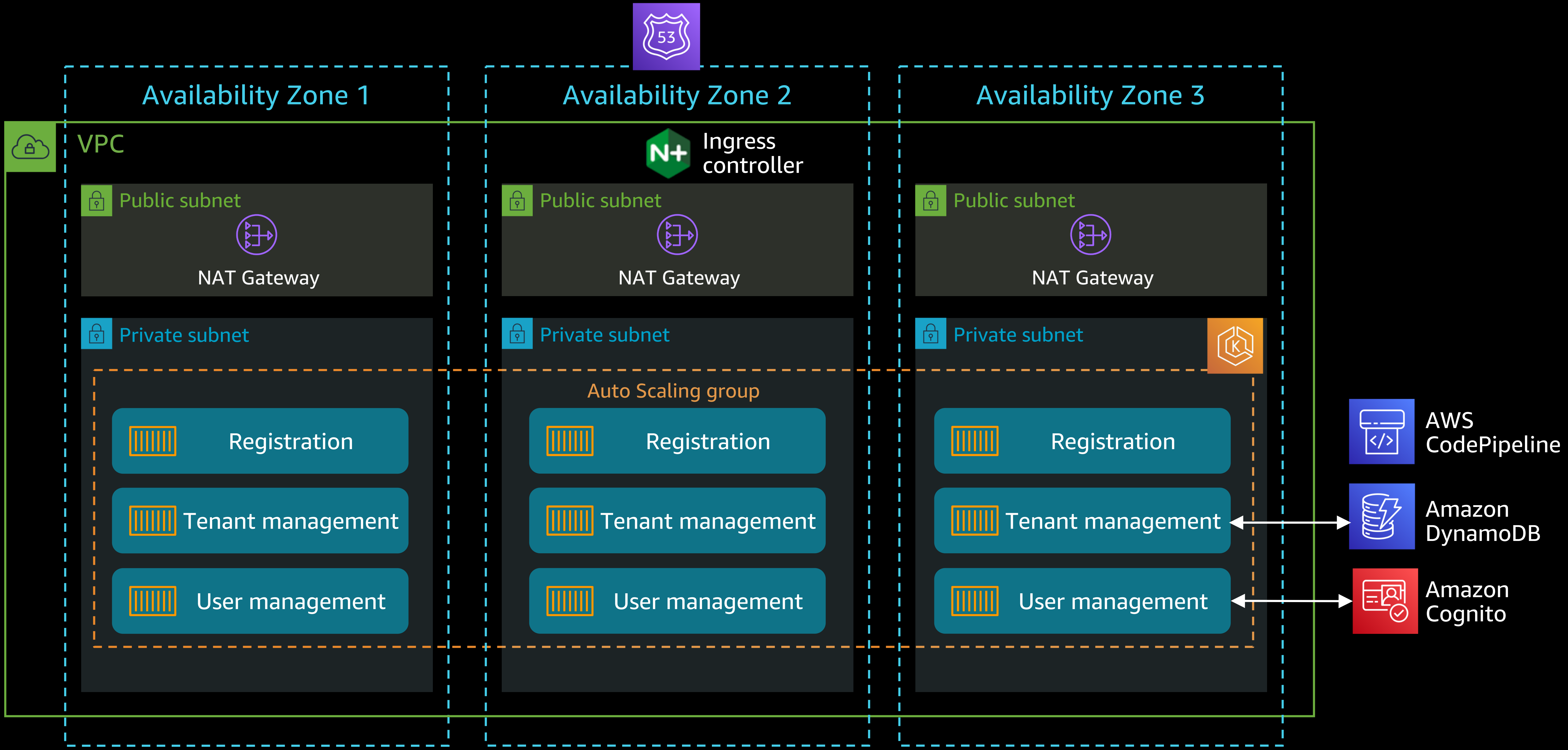
Amazon EKS SaaS architecture landscape



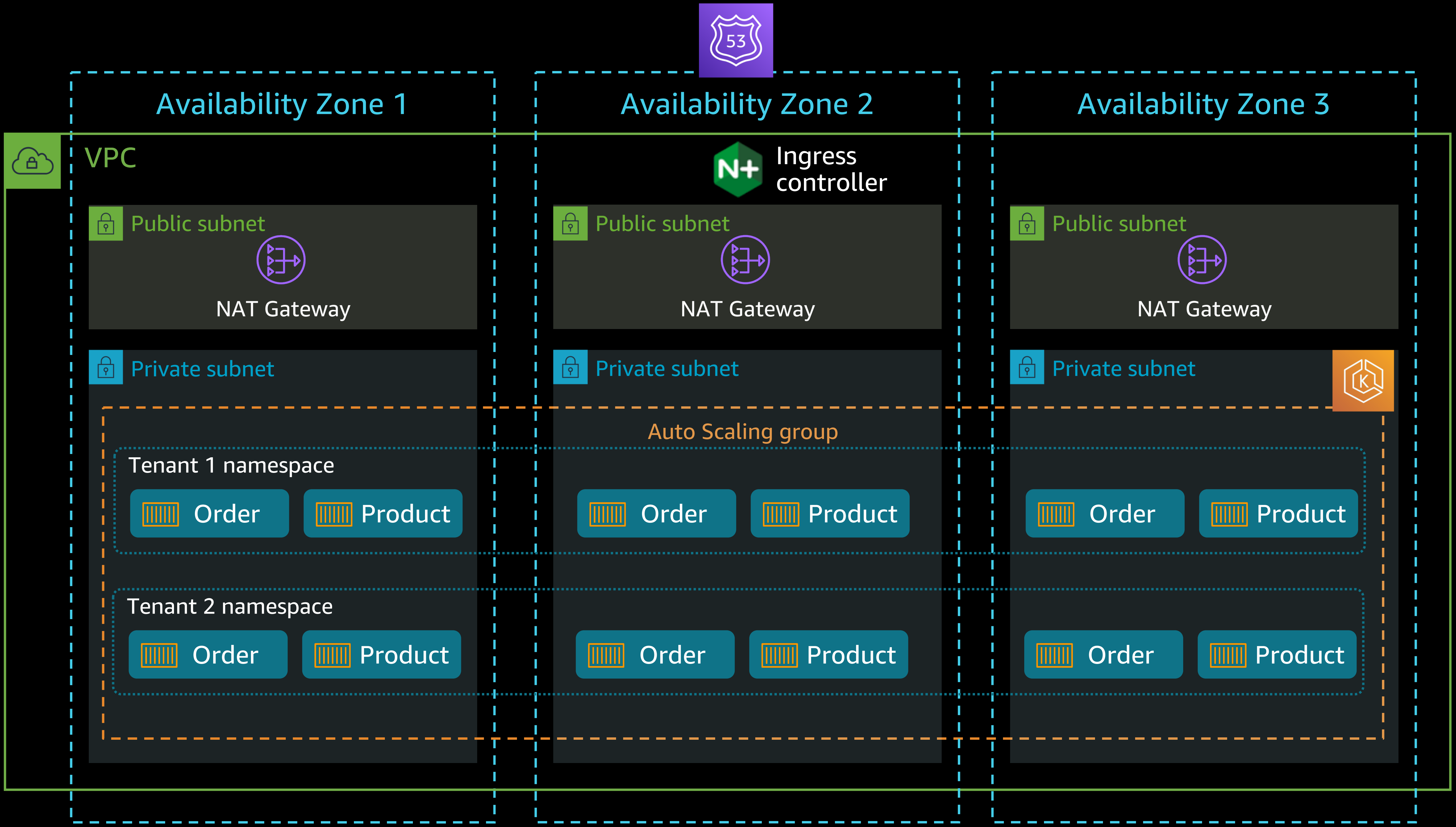
Conceptual view of the stack



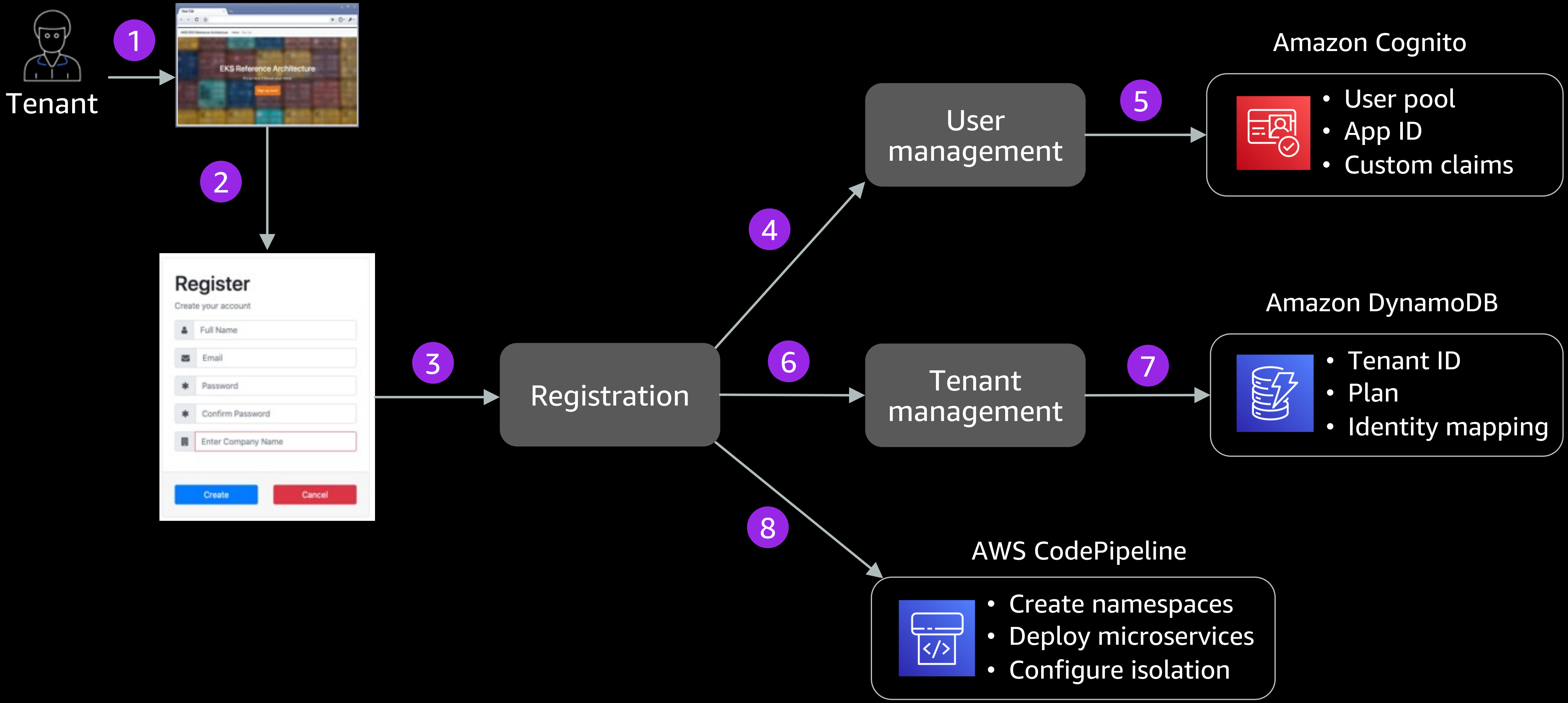
Provisioning the baseline environment



Tenant environments



Onboarding new tenants



Register

Create your account

Full Name

Email

Password

Confirm Password

Enter Company Name

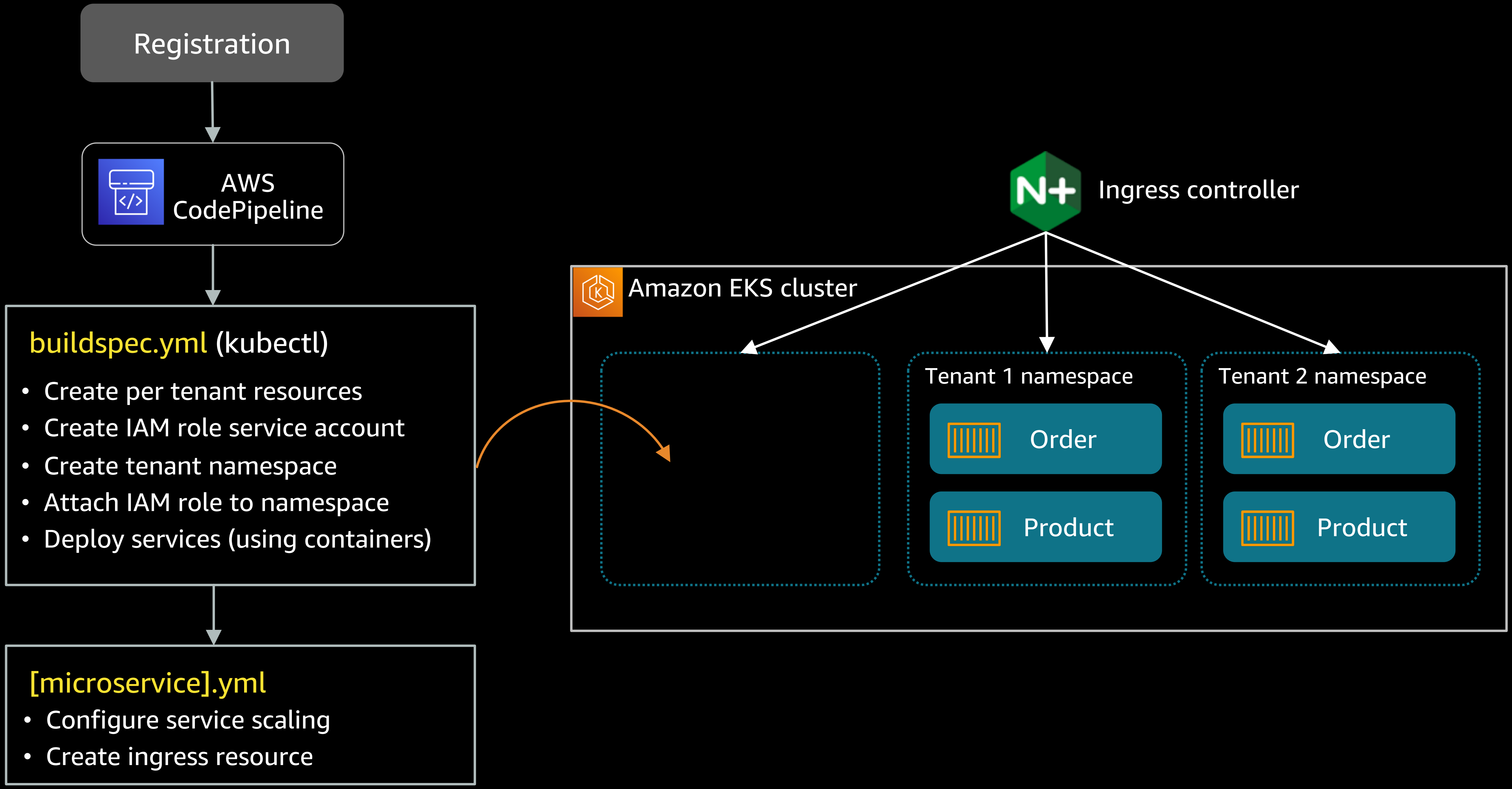
Create Cancel

Amazon Cognito
- User pool
- App ID
- Custom claims

Amazon DynamoDB
- Tenant ID
- Plan
- Identity mapping

AWS CodePipeline
- Create namespaces
- Deploy microservices
- Configure isolation

Tenant namespace provisioning



Registration

AWS CodePipeline

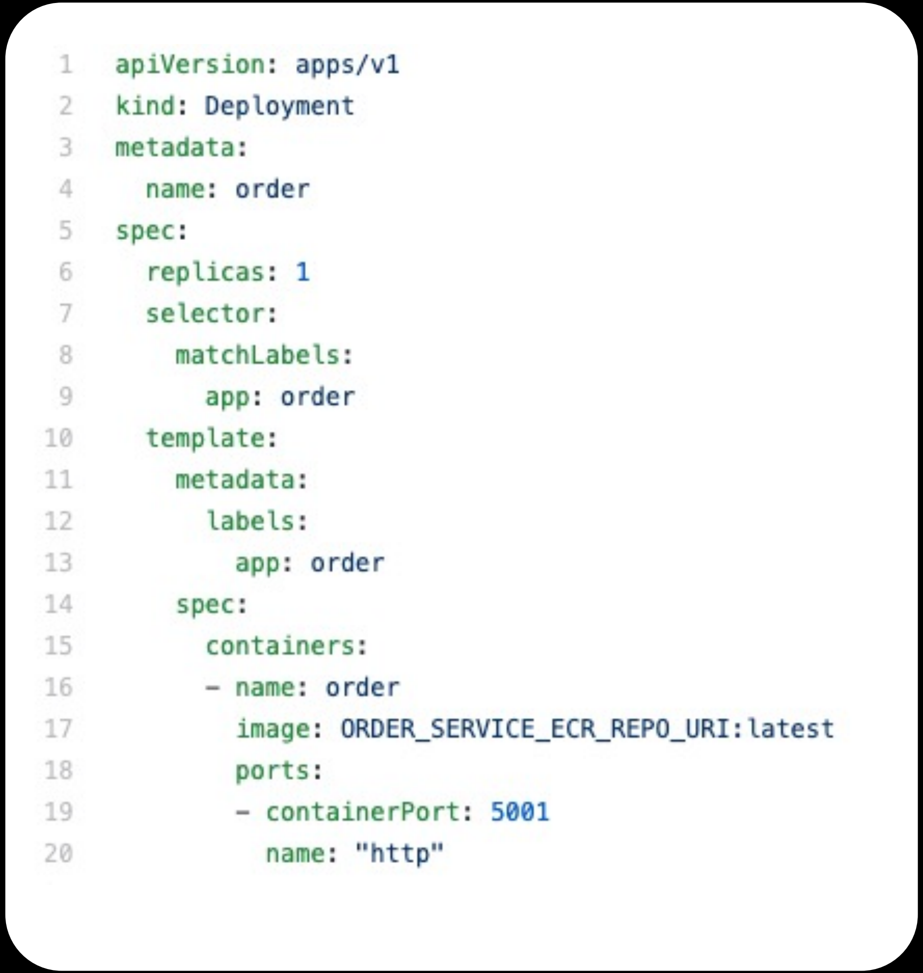
buildspec.yml (kubectl)

- Create per tenant resources
- Create IAM role service account
- Create tenant namespace
- Attach IAM role to namespace
- Deploy services (using containers)

[microservice].yml

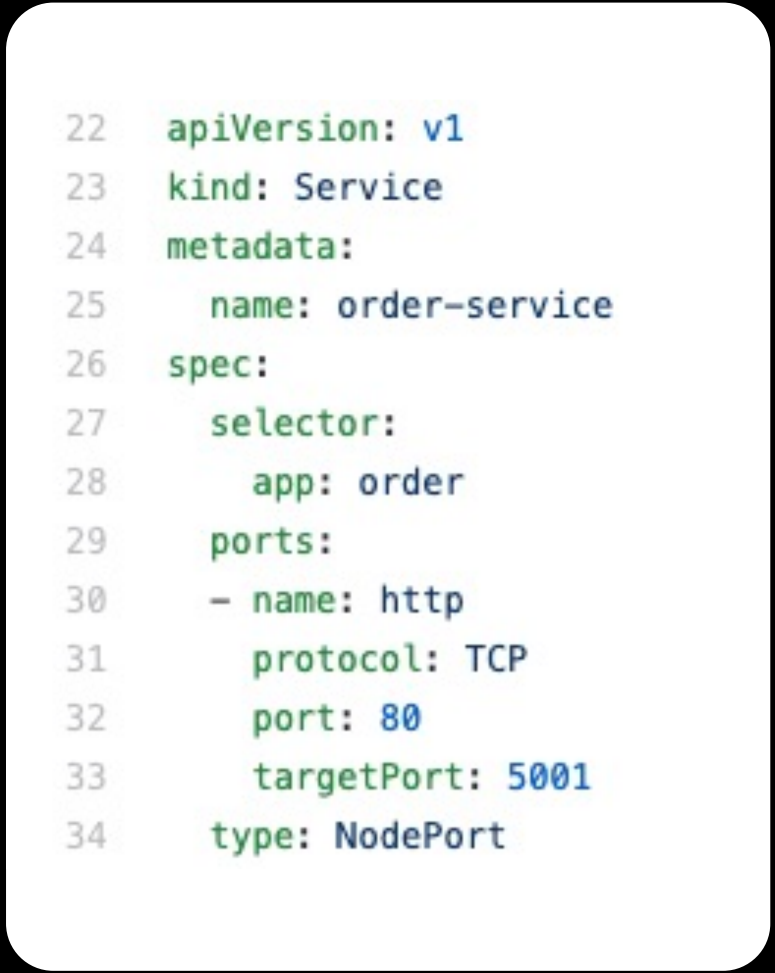
- Configure service scaling
- Create ingress resource

Microservice configuration



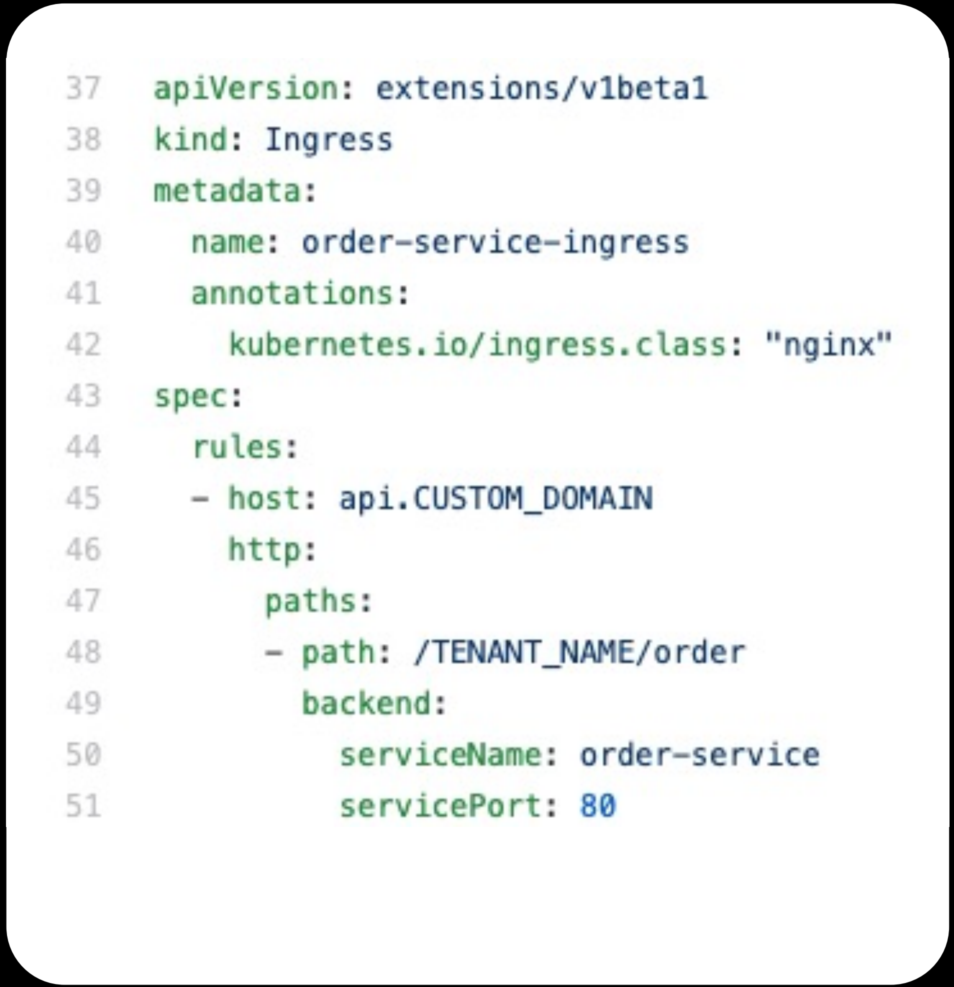
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order
  template:
    metadata:
      labels:
        app: order
    spec:
      containers:
      - name: order
        image: ORDER_SERVICE_ECR_REPO_URI:latest
        ports:
        - containerPort: 5001
          name: "http"
```

Deployment



```yaml
apiVersion: v1
kind: Service
metadata:
  name: order-service
spec:
  selector:
    app: order
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 5001
  type: NodePort
```

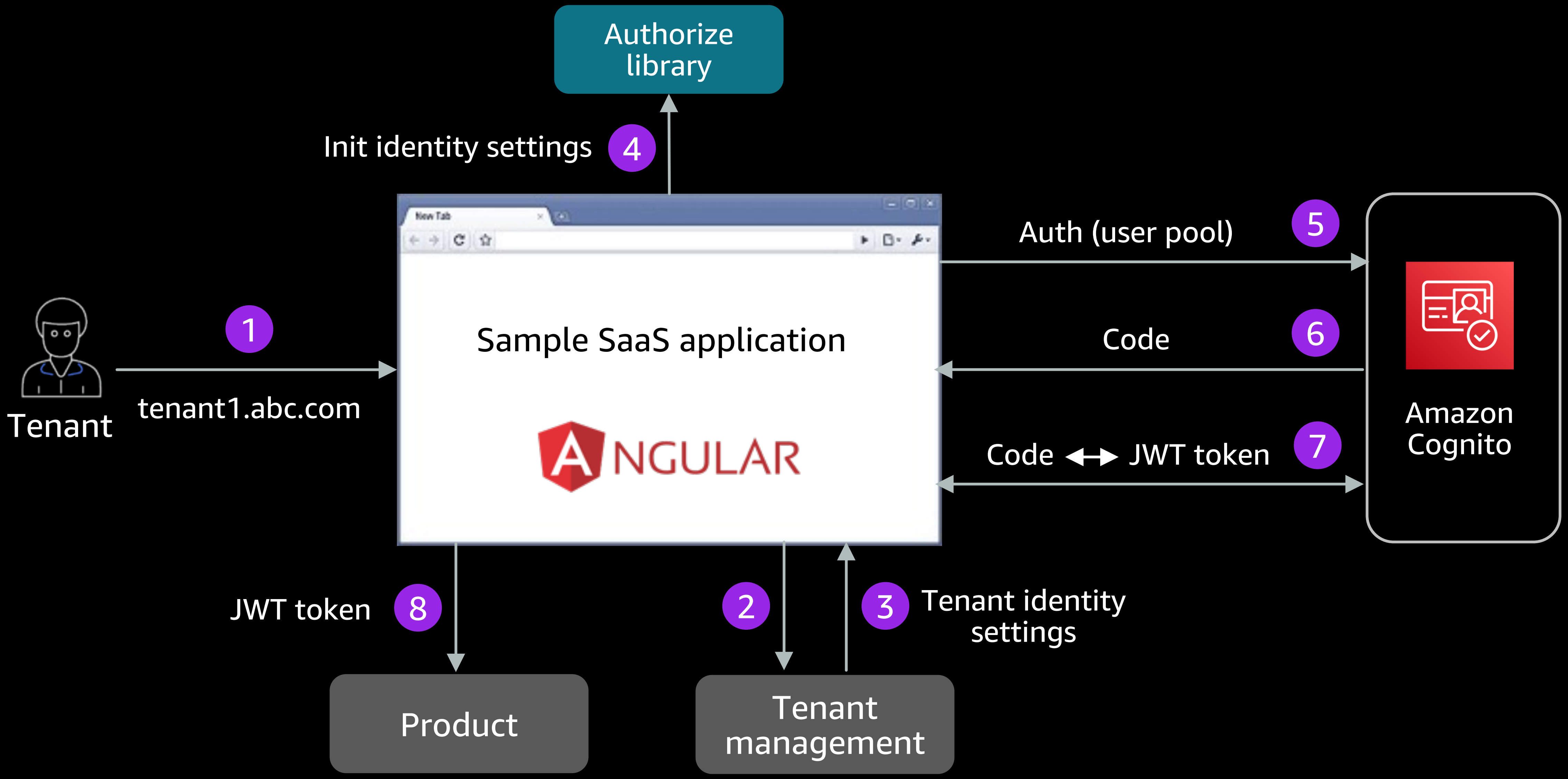
Service



```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: order-service-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: api.CUSTOM_DOMAIN
    http:
      paths:
      - path: /TENANT_NAME/order
        backend:
          serviceName: order-service
          servicePort: 80
```

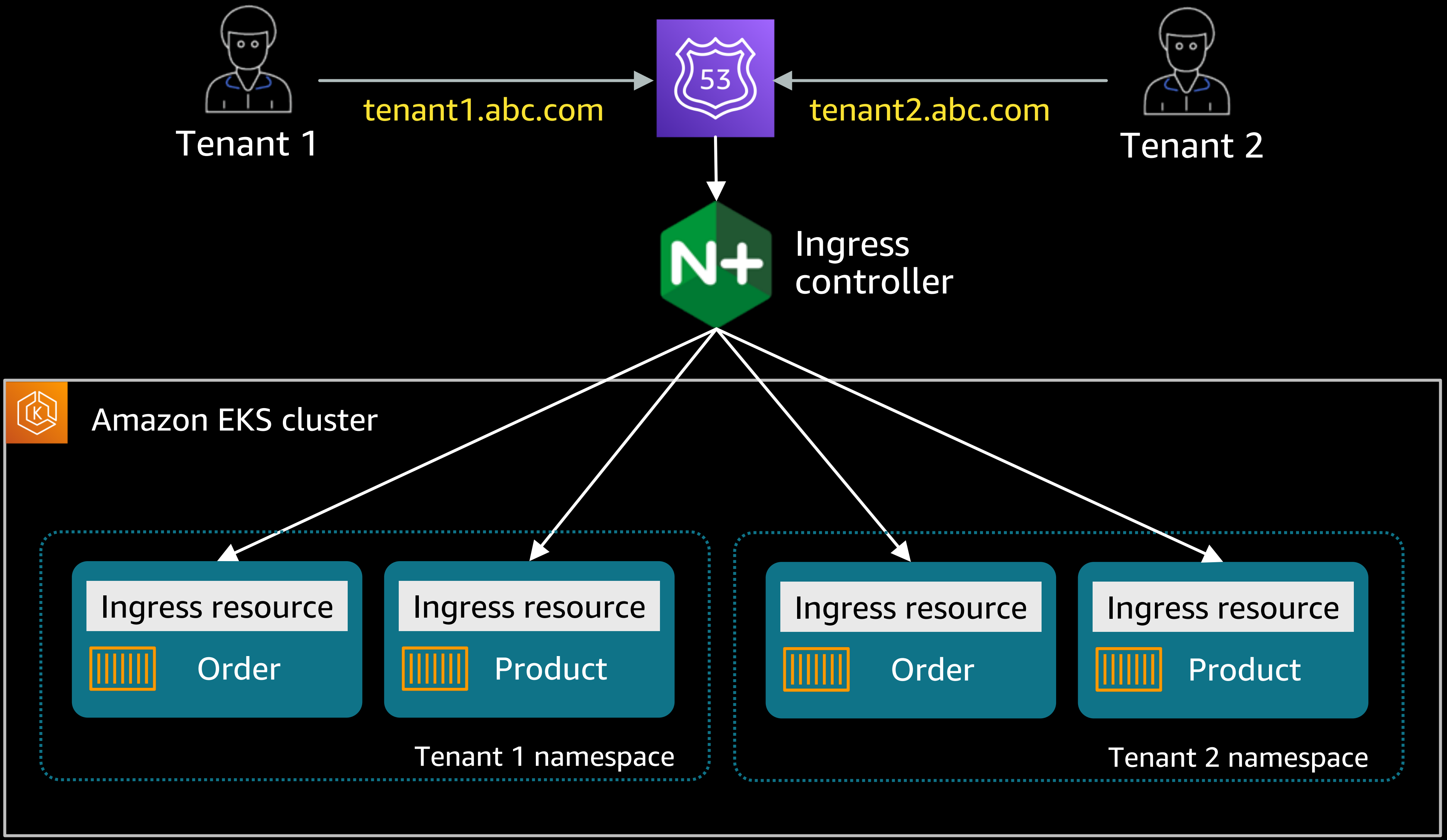
Ingress

Tenant authentication

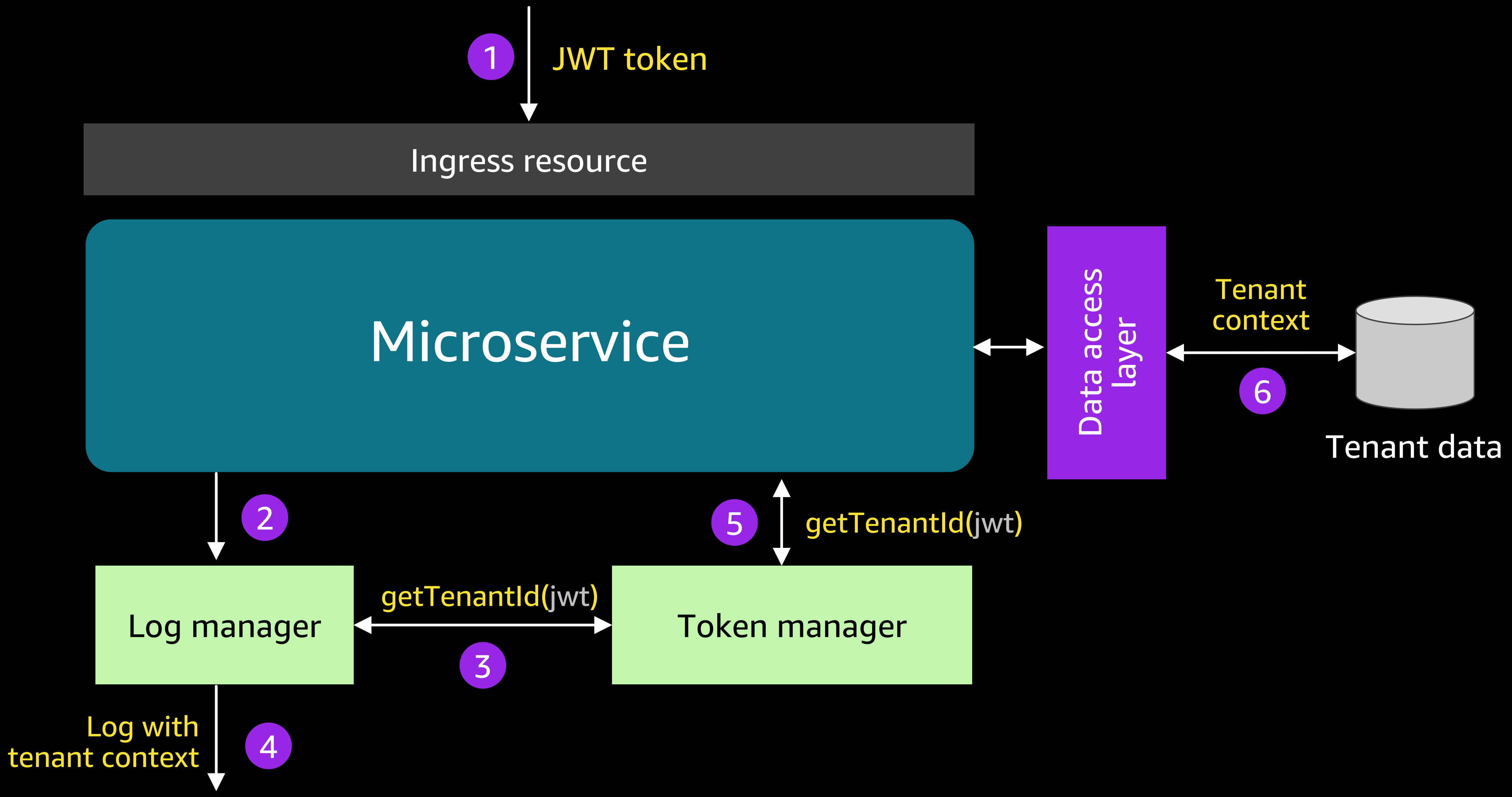


- Origin → company
- User pool mapping
- App ID

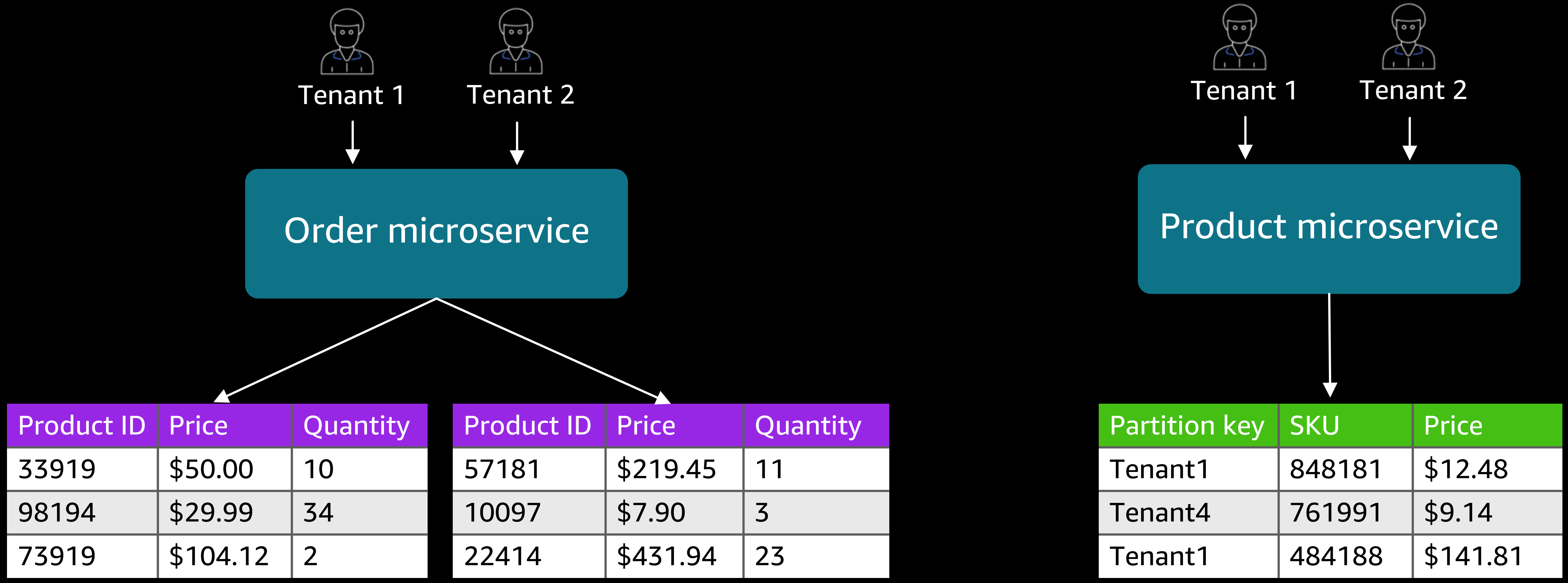
Routing tenants to namespaces



Inside the microservices



Data partitioning



Product ID	Price	Quantity
33919	$50.00	10
98194	$29.99	34
73919	$104.12	2

Order-Tenant1 table

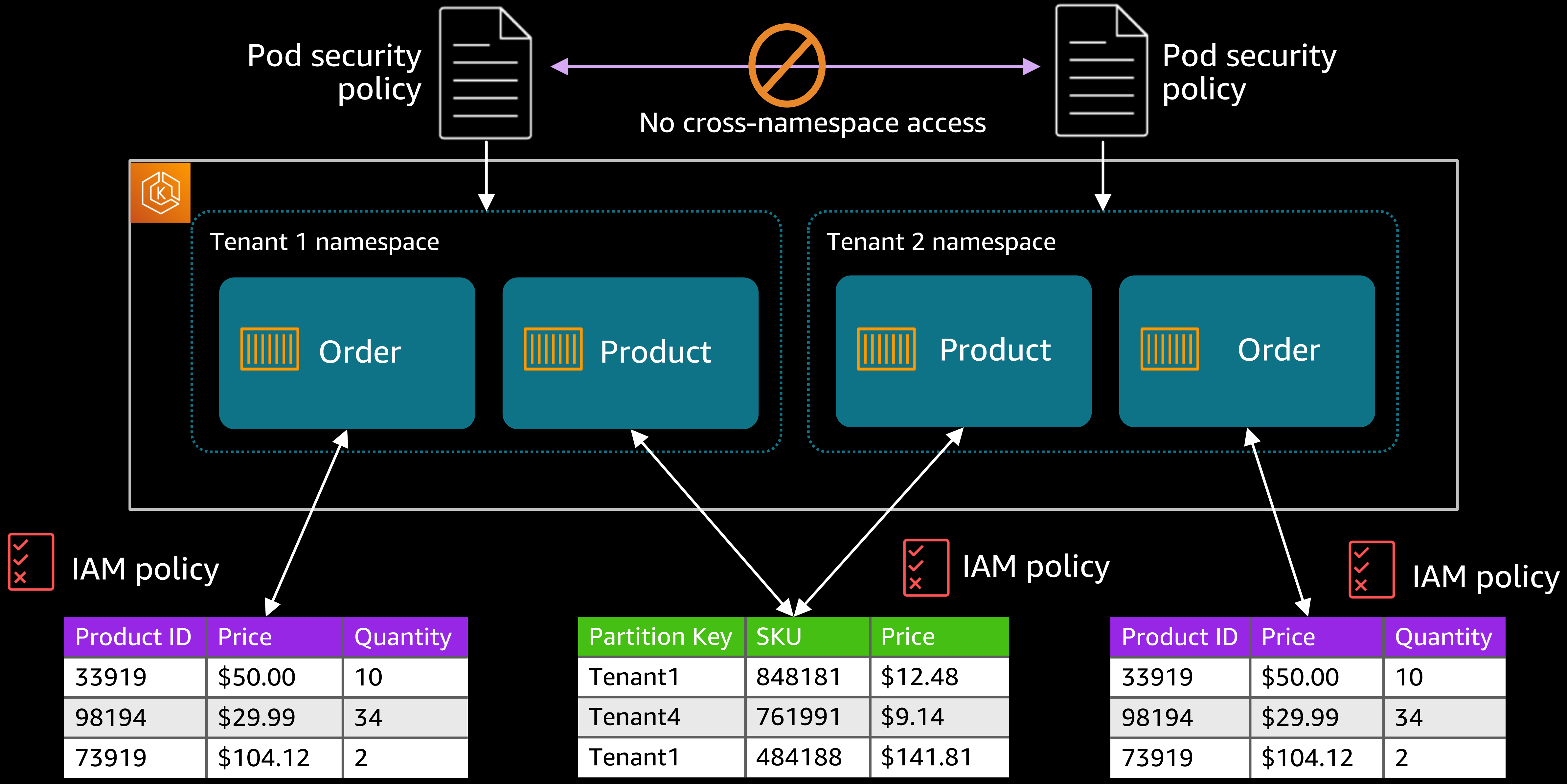
Product ID	Price	Quantity
57181	$219.45	11
10097	$7.90	3
22414	$431.94	23

Order-Tenant2 table

Partition key	SKU	Price
Tenant1	848181	$12.48
Tenant4	761991	$9.14
Tenant1	484188	$141.81

Product table

Tenant isolation



Product ID	Price	Quantity
33919	$50.00	10
98194	$29.99	34
73919	$104.12	2

Order-Tenant1 table

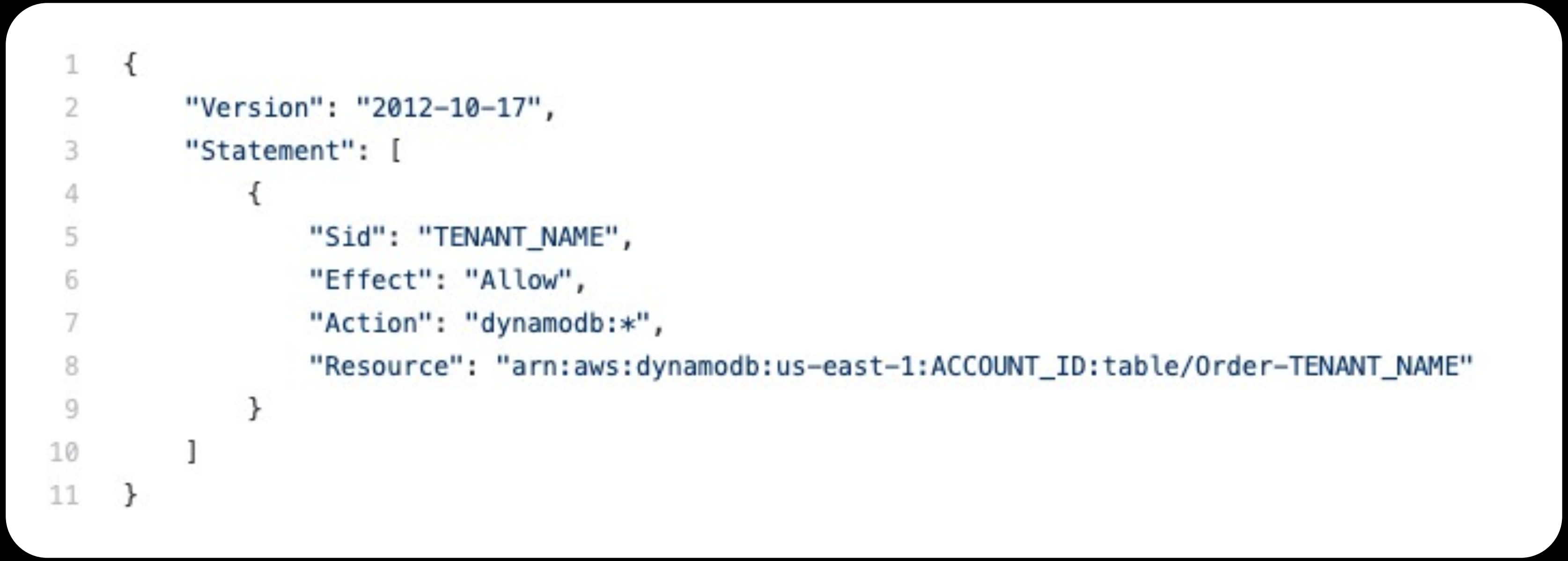
Partition Key	SKU	Price
Tenant1	848181	$12.48
Tenant4	761991	$9.14
Tenant1	484188	$141.81

Product table

Product ID	Price	Quantity
33919	$50.00	10
98194	$29.99	34
73919	$104.12	2

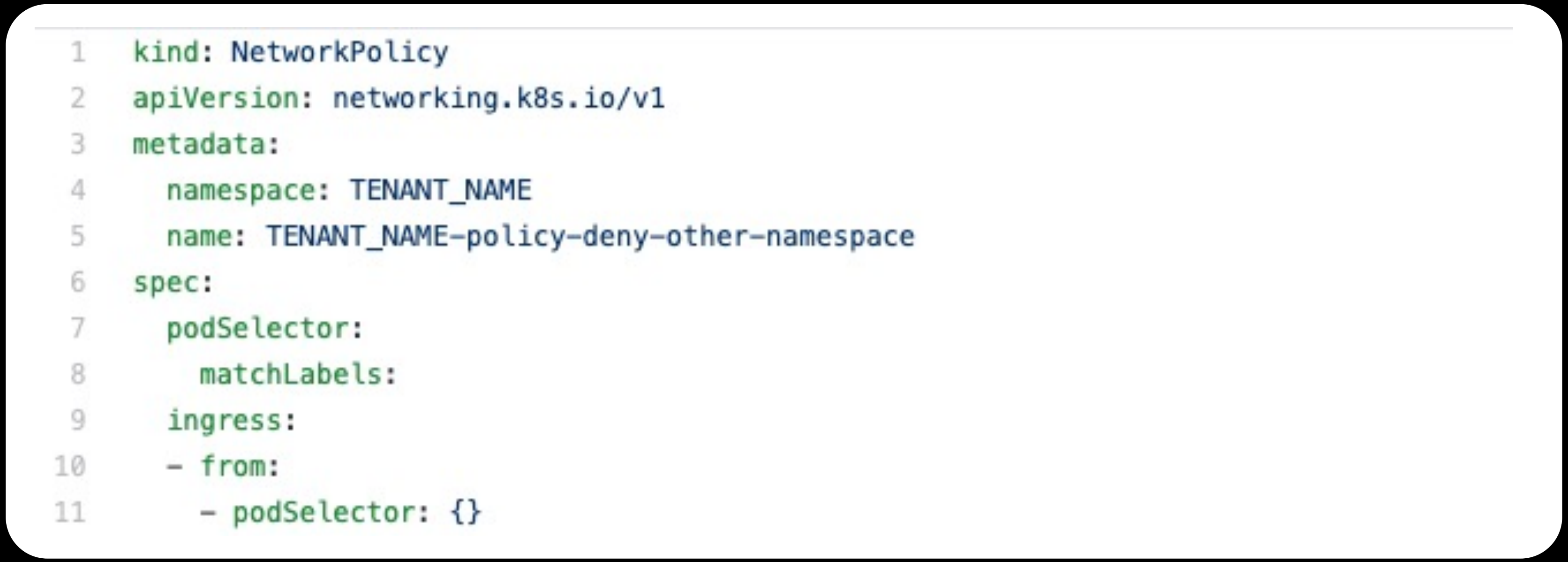
Order-Tenant2 table

Isolation policies



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TENANT_NAME",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-east-1:ACCOUNT_ID:table/Order-TENANT_NAME"
        }
    ]
}
```

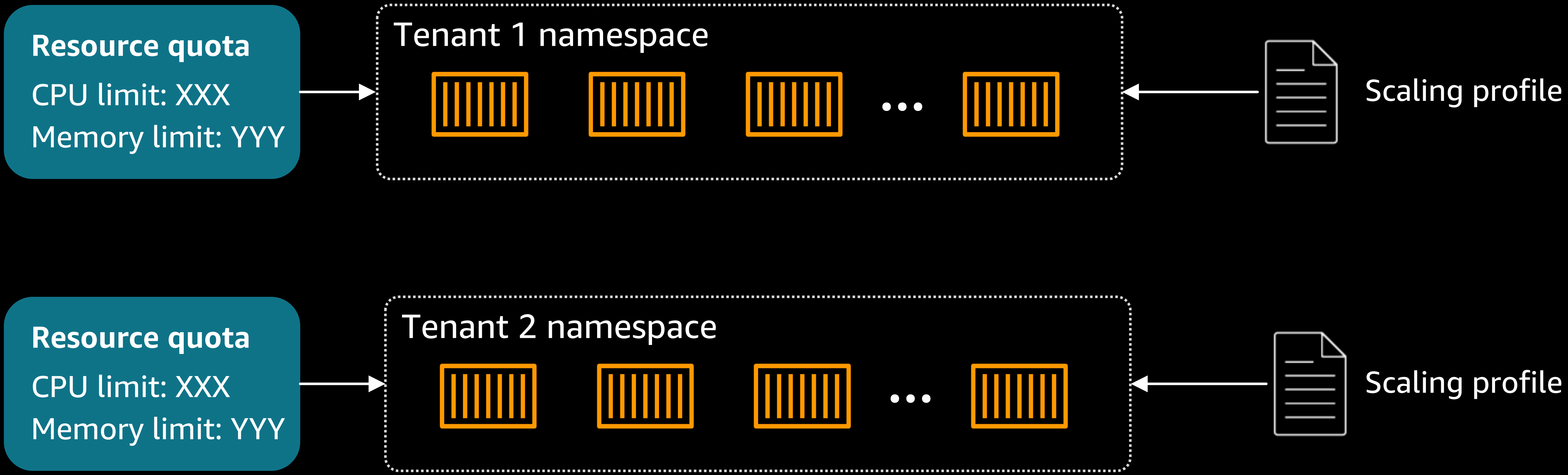
Scope access to order table by Tenant ID



```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: TENANT_NAME
  name: TENANT_NAME-policy-deny-other-namespace
spec:
  podSelector:
    matchLabels:
  ingress:
  - from:
    - podSelector: {}
```

Define network policy to limit access across namespaces

Scaling and tiering considerations



Resource quota

CPU limit: XXX

Memory limit: YYY

Tenant 1 namespace

Scaling profile

Resource quota

CPU limit: XXX

Memory limit: YYY

Tenant 2 namespace

Scaling profile

SaaS multi-tenant storage strategies

The idea behind data partitioning

Tenant 1

Tenant 2

Tenant 3

Tenant 4

Product data

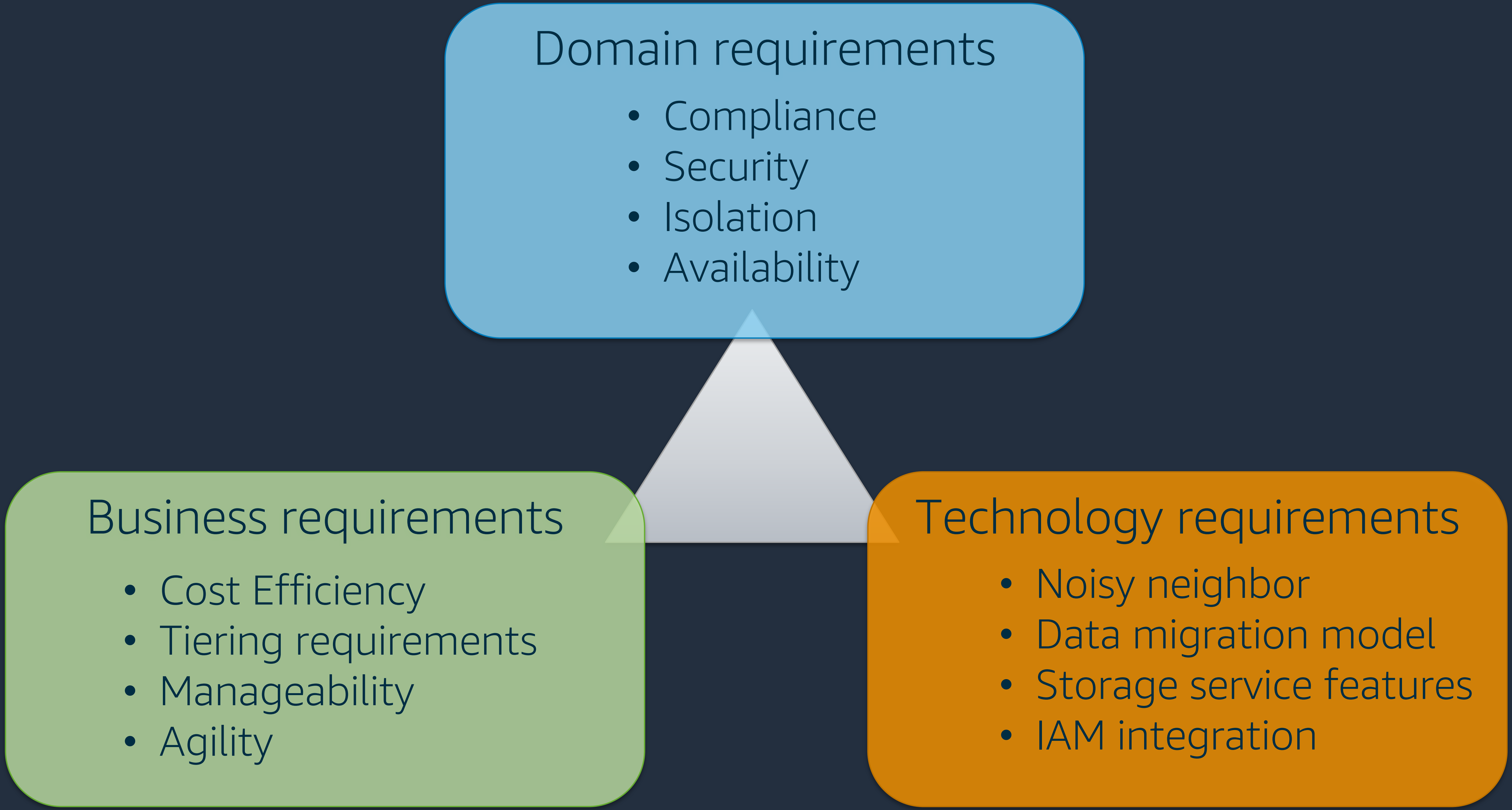
Product Id	Name	SKU	Cost
1940-939-94	Glove	939301	12.39
3538-819-11	Shirt	194193	7.83
1464-992-12	Hat	539294	15.41
8810-098-53	Scarf	793891	130.84

Order data

Oder Id	Product Id	Qty
9314-114-91	1940-939-94	1
7544-325-98	8810-098-53	4
8755-069-24	1940-939-94	2
4991-630-04	3538-819-11	1

- Which data belongs to which tenants?
- How is data accessed in the context of each tenant?
- What are the implications that are associated with different partitioning models?

Many factors shape your partitioning strategy



Domain requirements

- Compliance
- Security
- Isolation
- Availability

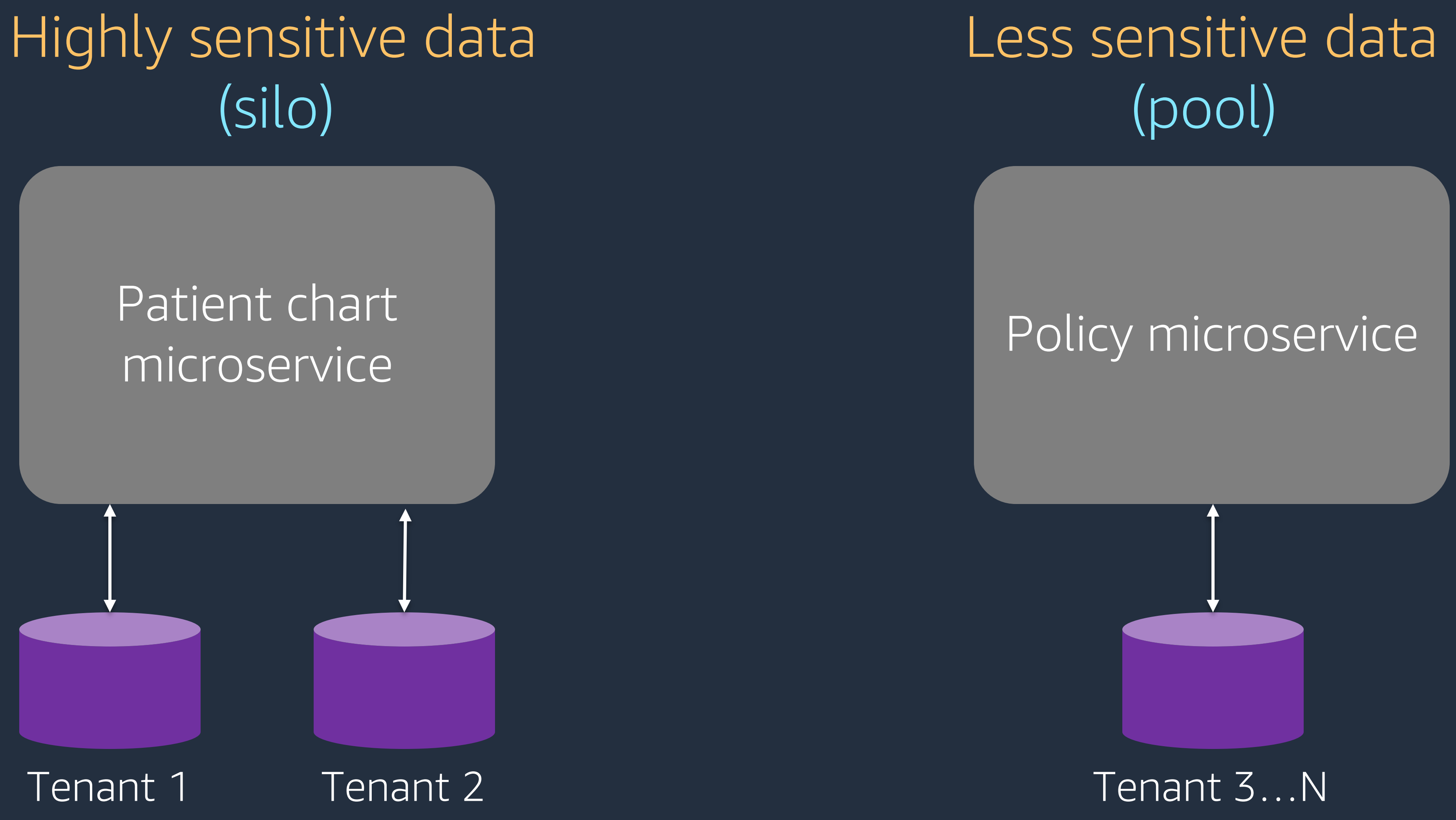
Business requirements

- Cost Efficiency
- Tiering requirements
- Manageability
- Agility

Technology requirements

- Noisy neighbor
- Data migration model
- Storage service features
- IAM integration

Compliance and data partitioning



Highly sensitive data

(silo)

Patient chart microservice

Tenant 1

Tenant 2

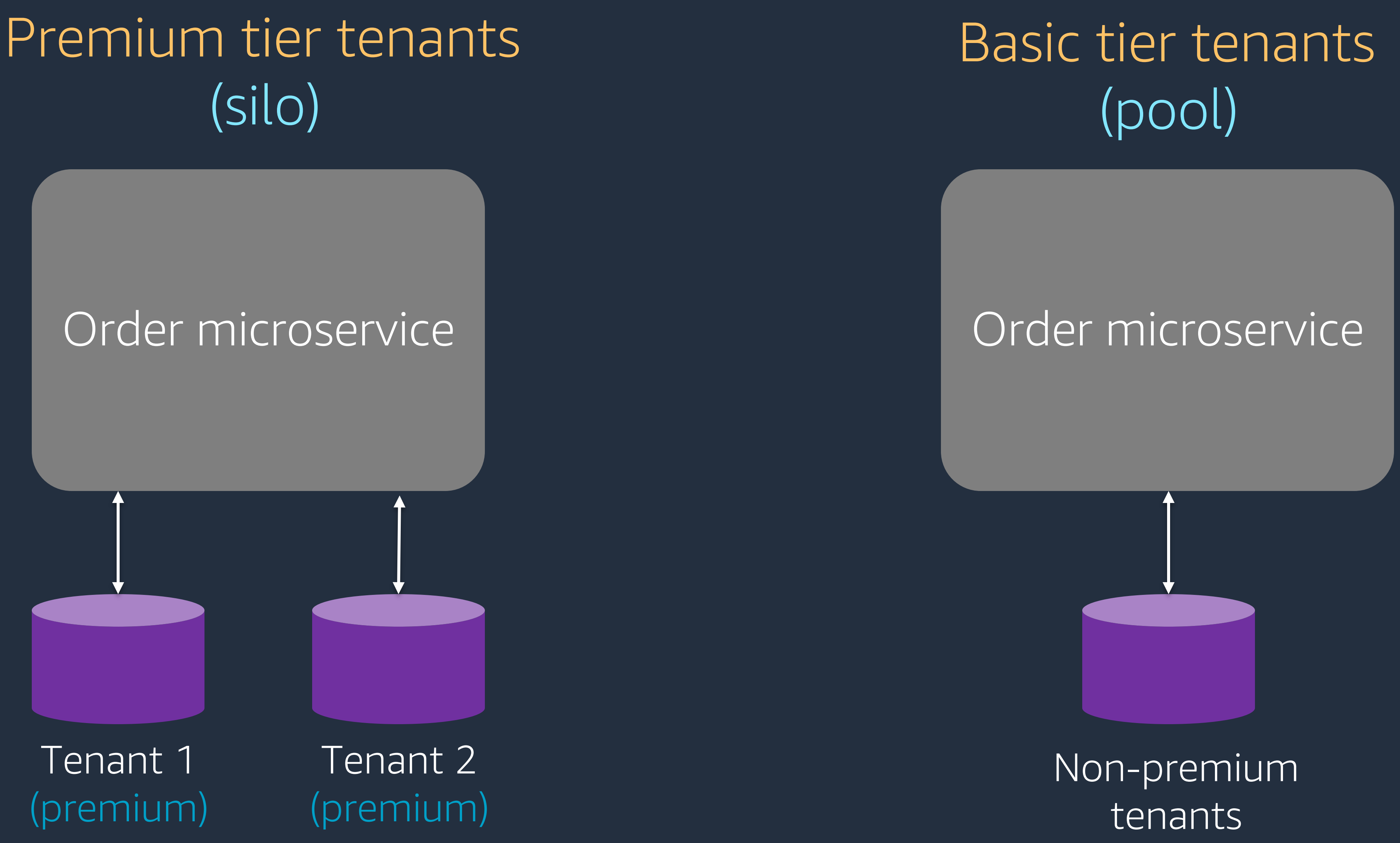
Less sensitive data

(pool)

Policy microservice

Tenant 3…N

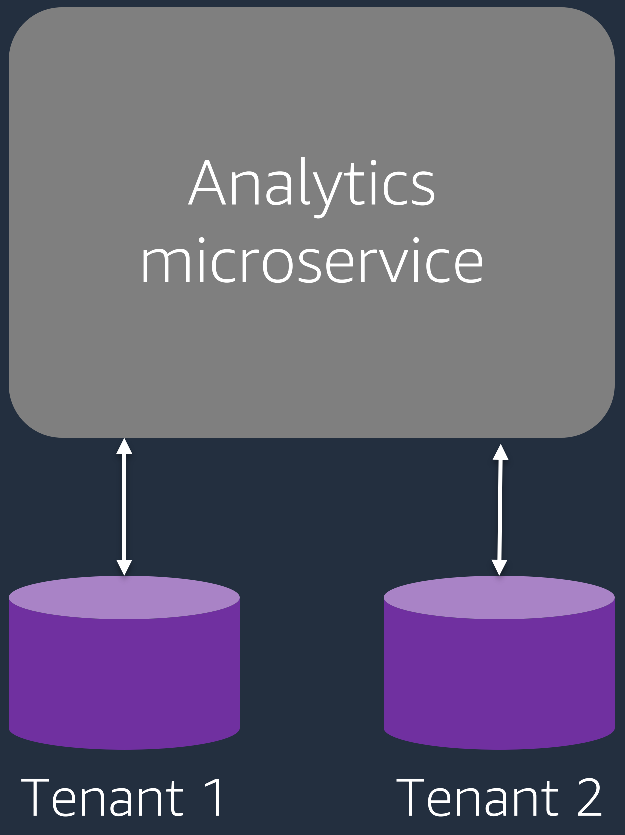
Tiering can influence partitioning



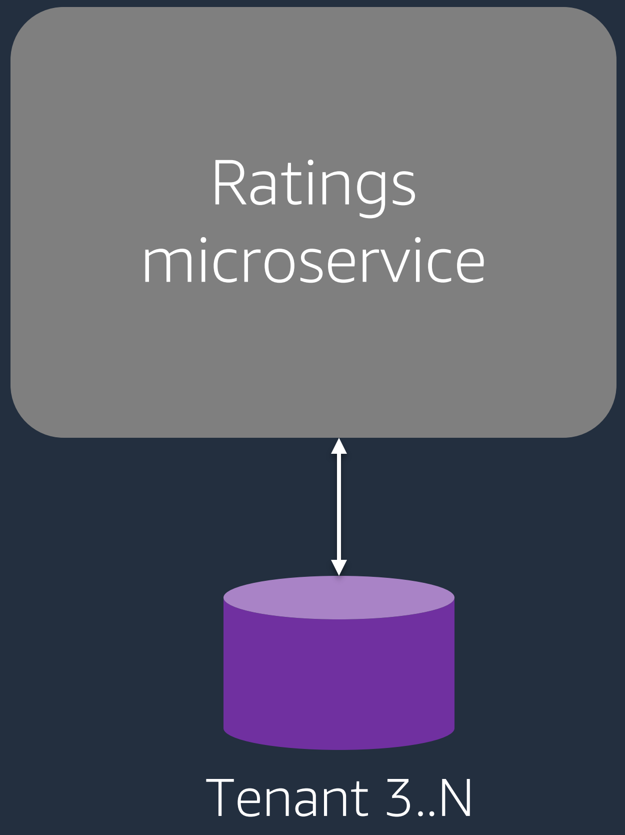
Workload driven data partitioning

Business critical use cases (silo)

- Volume
- Complexity
- Data size
- SLAs
- Noisy neighbor



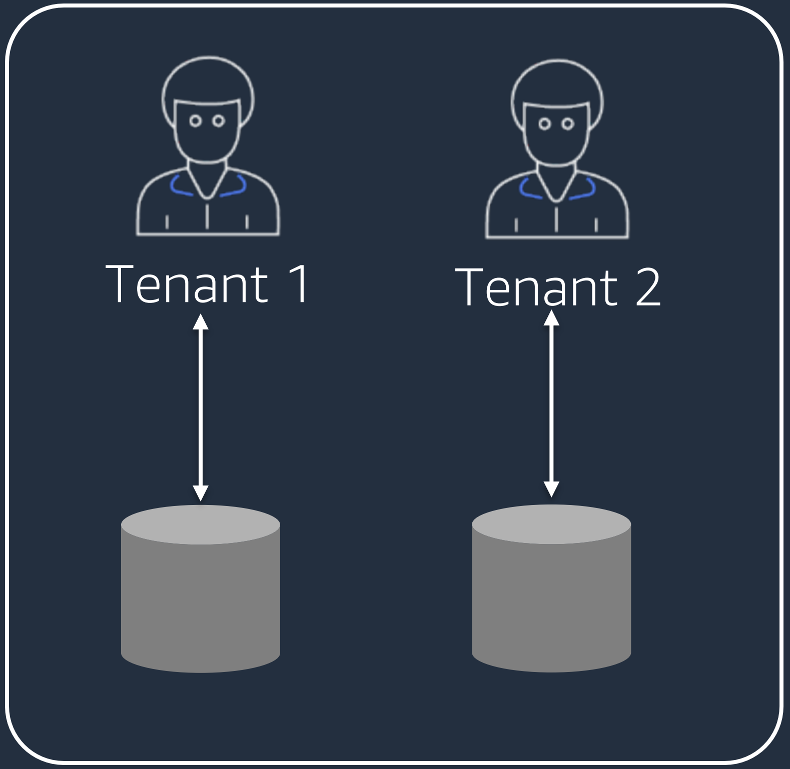
Low impact use cases (pool)



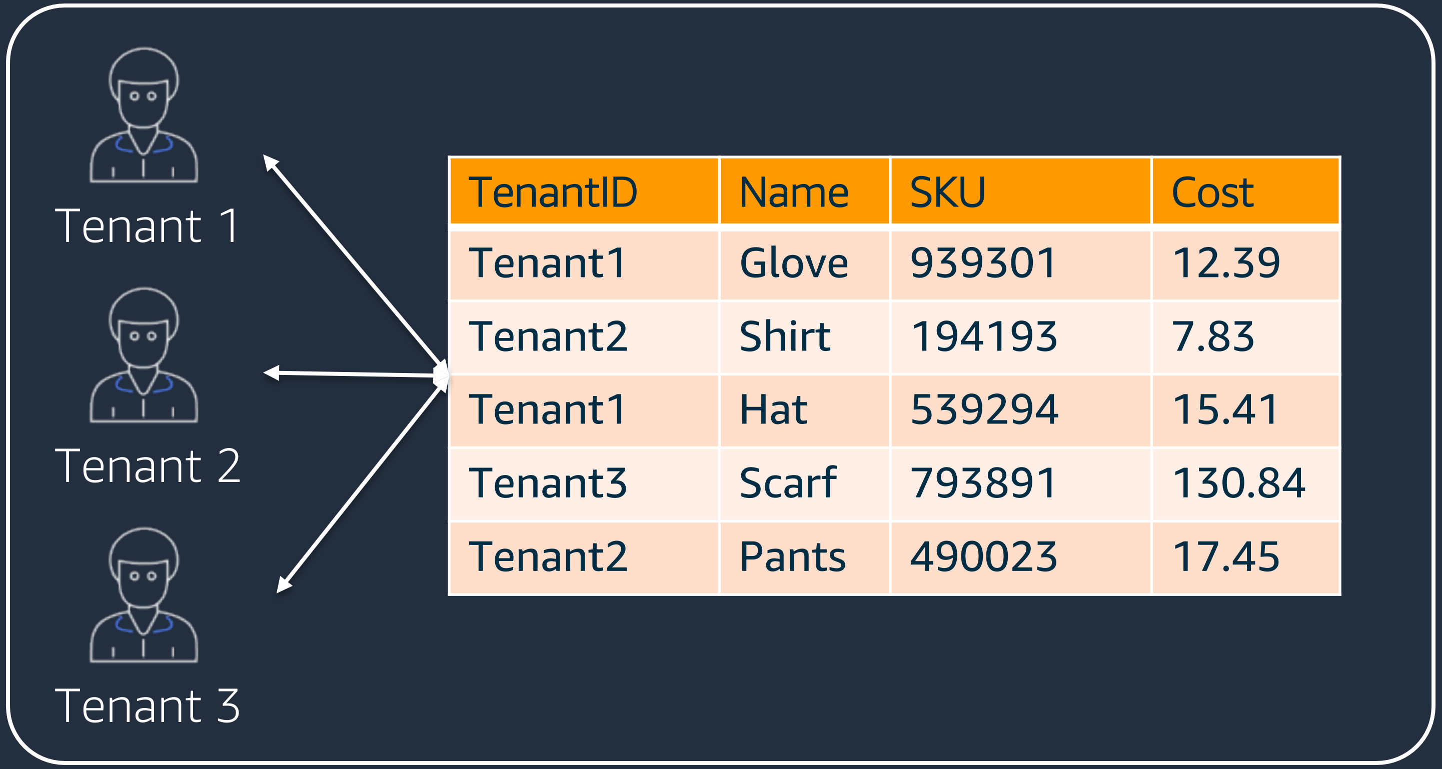
- Infrequent access
- Small data

Two primary models for partitioning data

Silo model



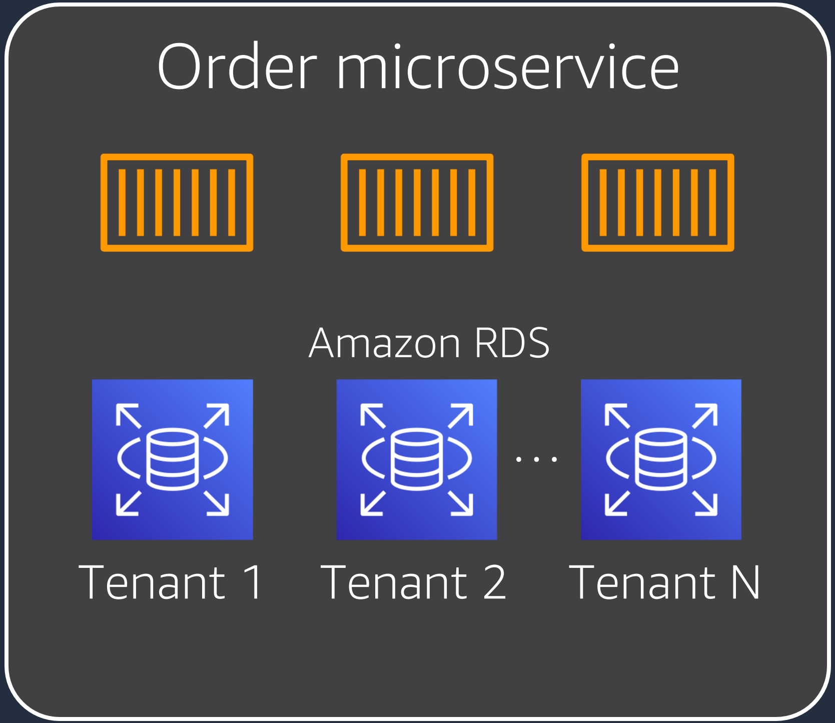
Pool model



TenantID	Name	SKU	Cost
Tenant1	Glove	939301	12.39
Tenant2	Shirt	194193	7.83
Tenant1	Hat	539294	15.41
Tenant3	Scarf	793891	130.84
Tenant2	Pants	490023	17.45

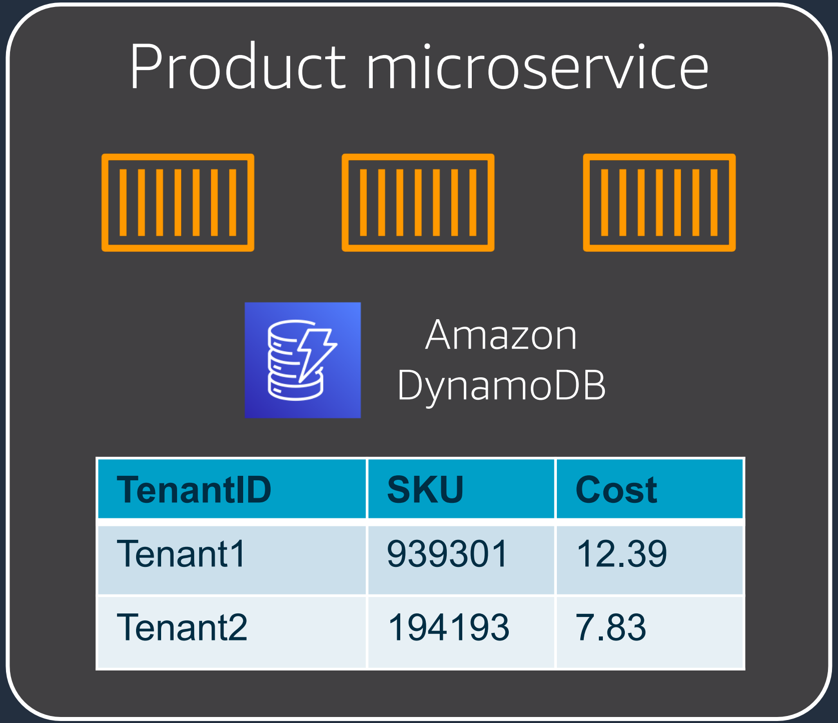
Not a one-size-fits-all model

Partitioning and isolation strategy should be decided on a service-by-service basis



Order microservice

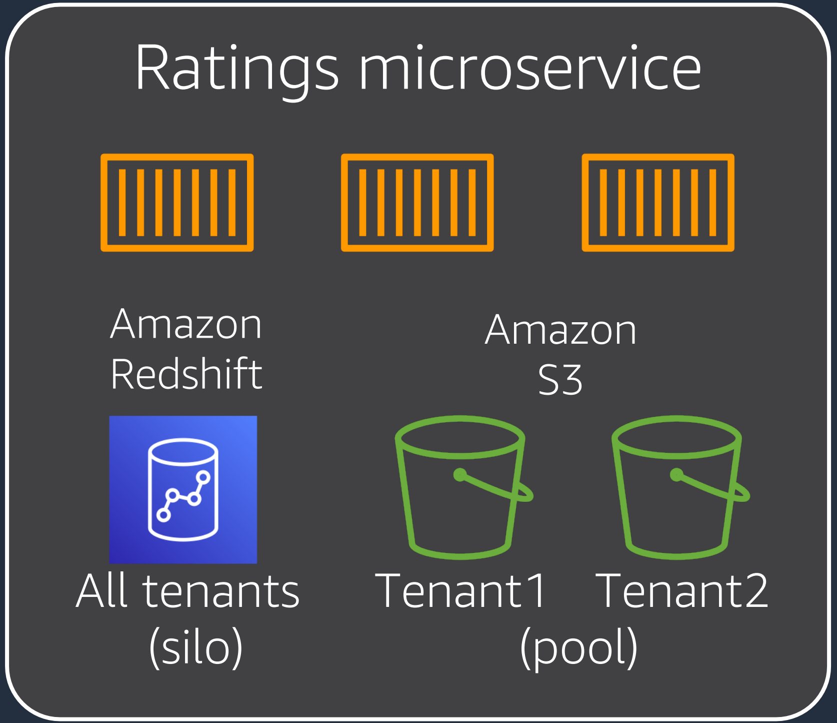
Database per tenant (silo)



Product microservice

TenantID	SKU	Cost
Tenant1	939301	12.39
Tenant2	194193	7.83

Shared database for all tenants (pool)



Ratings microservice

Mixed mode - silo and pool models

IAM granularity can be a factor

Course-grained IAM control

Amazon RDS

Amazon Elasticsearch Service

Fine-grained IAM control

Amazon DynamoDB

Amazon Simple Storage Service (S3)

A different strategy for each service

Amazon RDS

Amazon DynamoDB

Amazon Elasticsearch Service

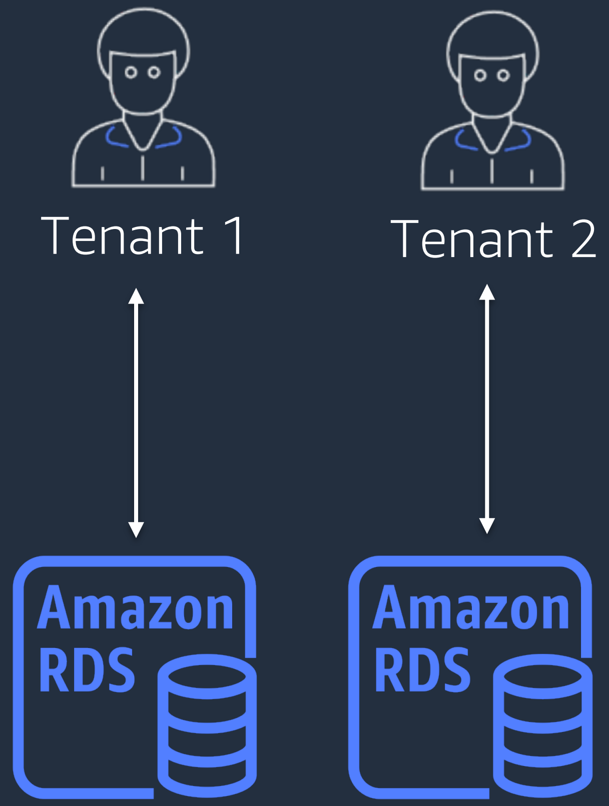
Amazon Simple Storage Service (S3)

Amazon Timestream

Amazon Redshift

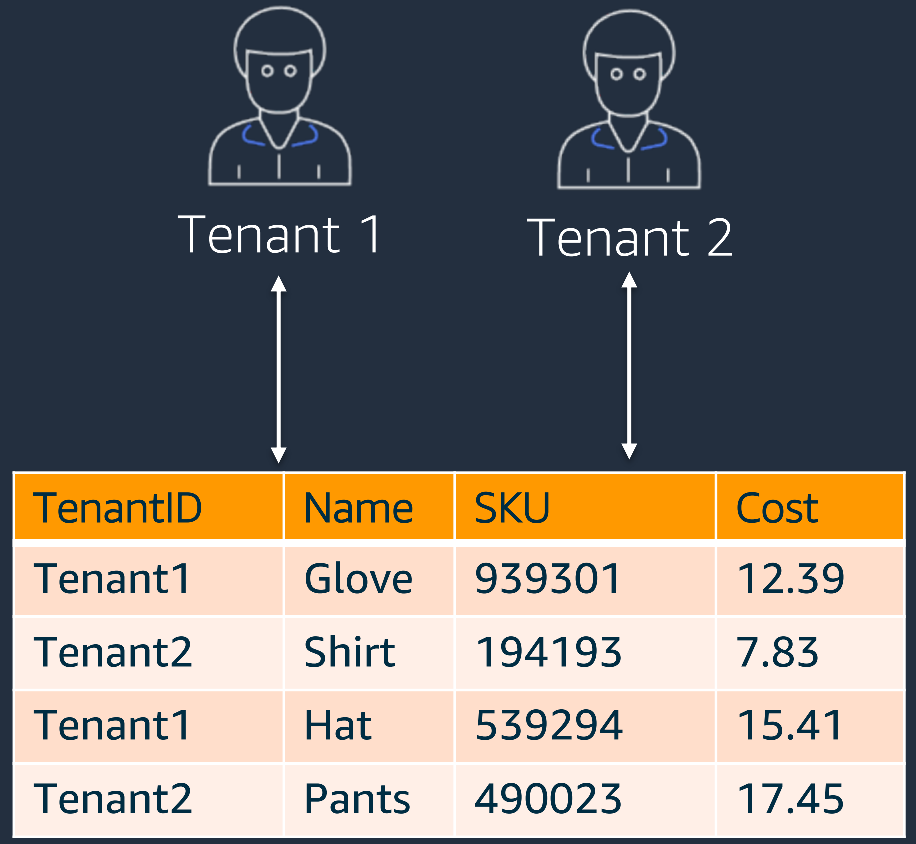
Data partitioning with RDS

Silo model



Separate instance/database per tenant

Pool model

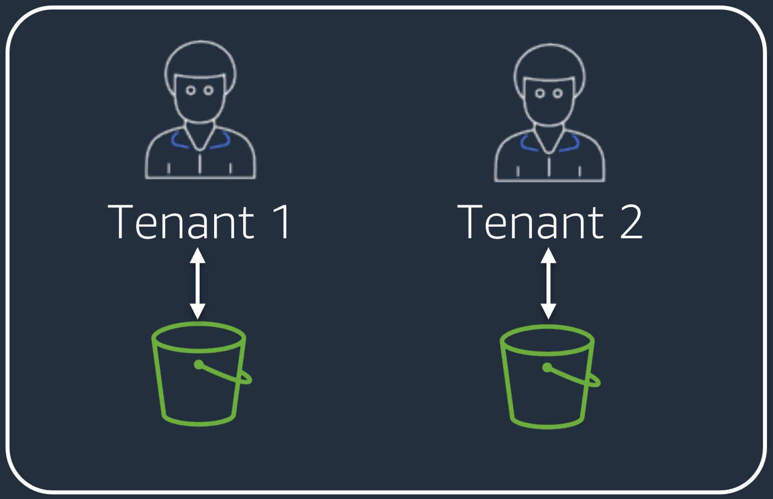


TenantID	Name	SKU	Cost
Tenant1	Glove	939301	12.39
Tenant2	Shirt	194193	7.83
Tenant1	Hat	539294	15.41
Tenant2	Pants	490023	17.45

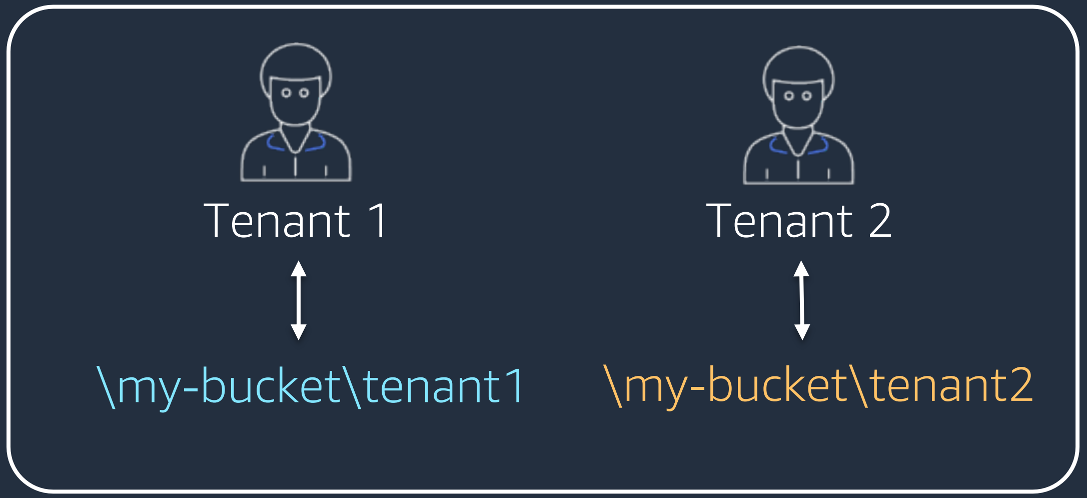
One database with tenant-indexed data

Data partitioning with S3

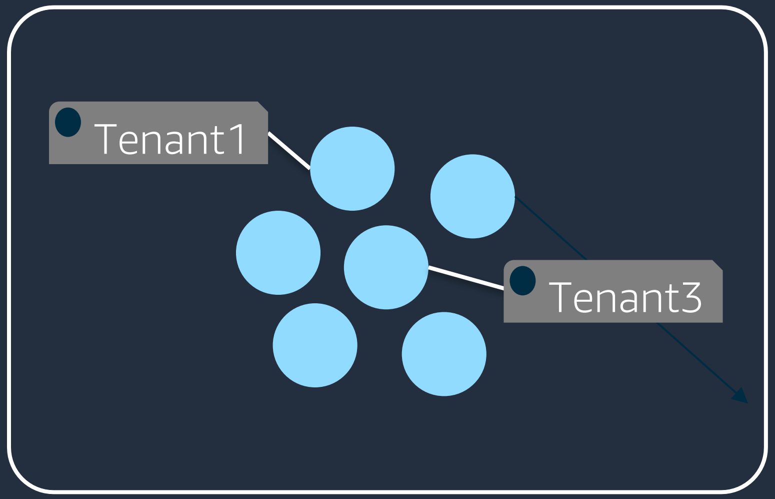
Bucket per tenant



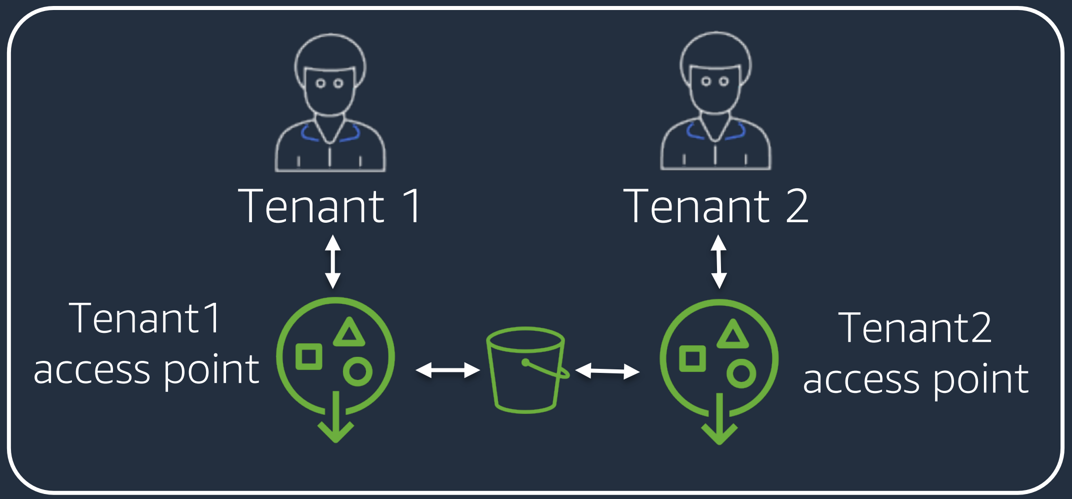
Prefix per tenant



Tag per tenant



Access point per tenant



Data partitioning with Amazon DynamoDB

Table per tenant (silo)

Tenant 1

Name	SKU	Cost
Glove	939301	12.39
Shirt	194193	7.83
Hat	539294	15.41
Pants	490023	17.45

order-table-tenant1

Tenant 2

Name	SKU	Cost
Shoes	840842	15.49
Hat	491309	44.89
Scarf	112514	96.01
Socks	309194	83.35

order-table-tenant2

Shared tenant table (pool)

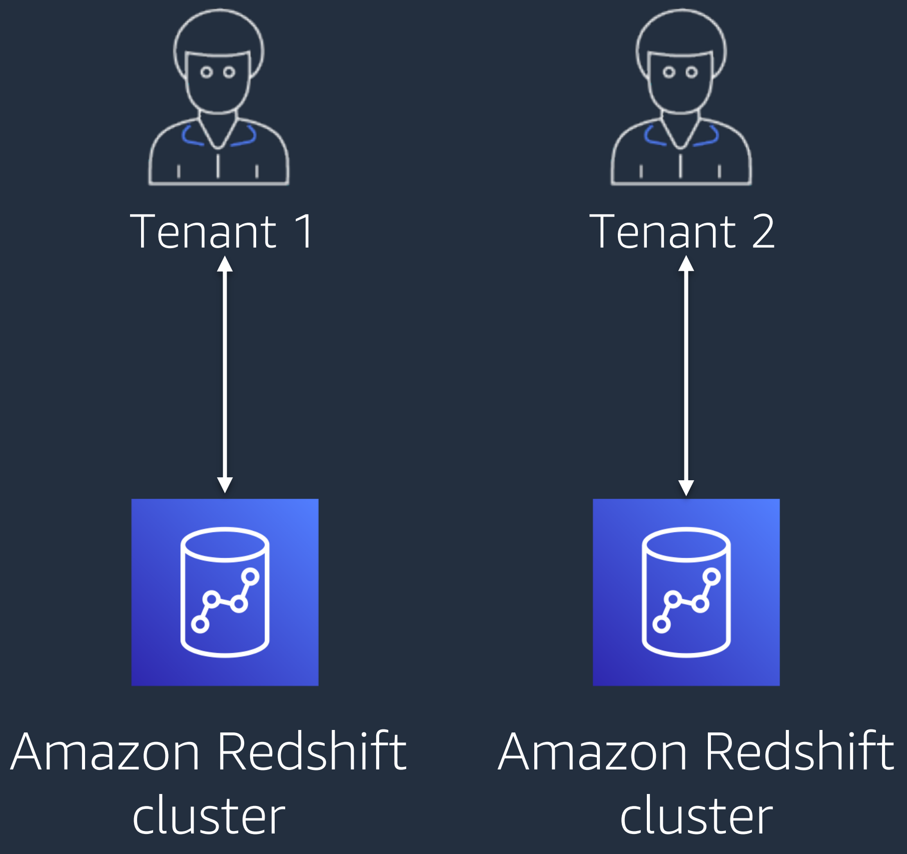
Tenant 1 Tenant 2

TenantID	Name	SKU	Cost
Tenant1	Hat	230914	7.35
Tenant2	Socks	581641	15.98
Tenant1	Gloves	402198	89.43
Tenant2	Shirt	620985	17.64

order-table

Data partitioning with Redshift

Cluster per tenant (silo)



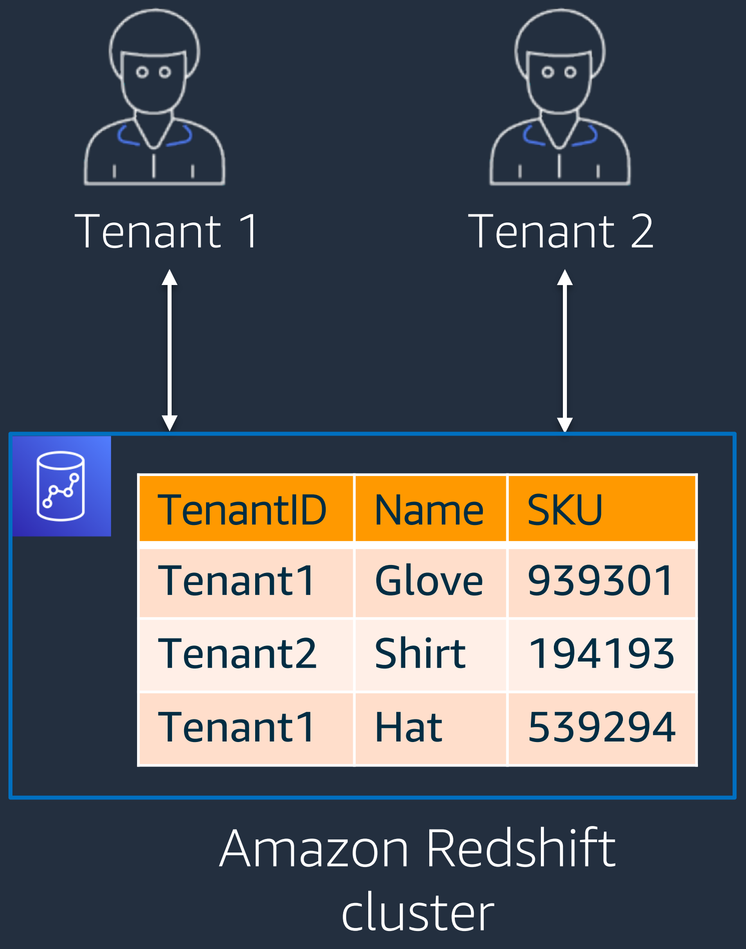
Tenant 1

Tenant 2

Amazon Redshift cluster

Amazon Redshift cluster

Shared cluster for all tenants (pool)



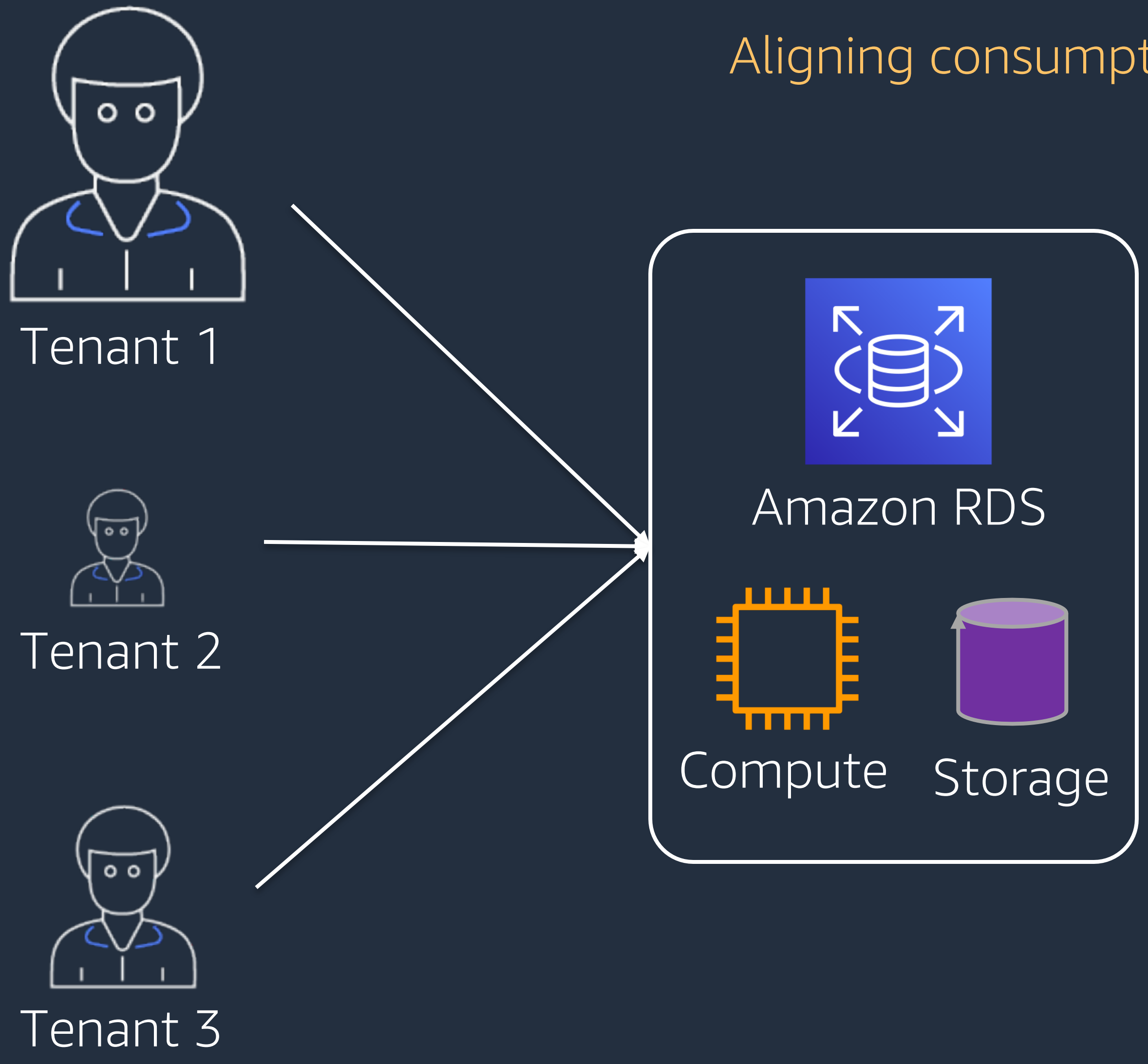
Tenant 1

Tenant 2

TenantID	Name	SKU
Tenant1	Glove	939301
Tenant2	Shirt	194193
Tenant1	Hat	539294

Amazon Redshift cluster

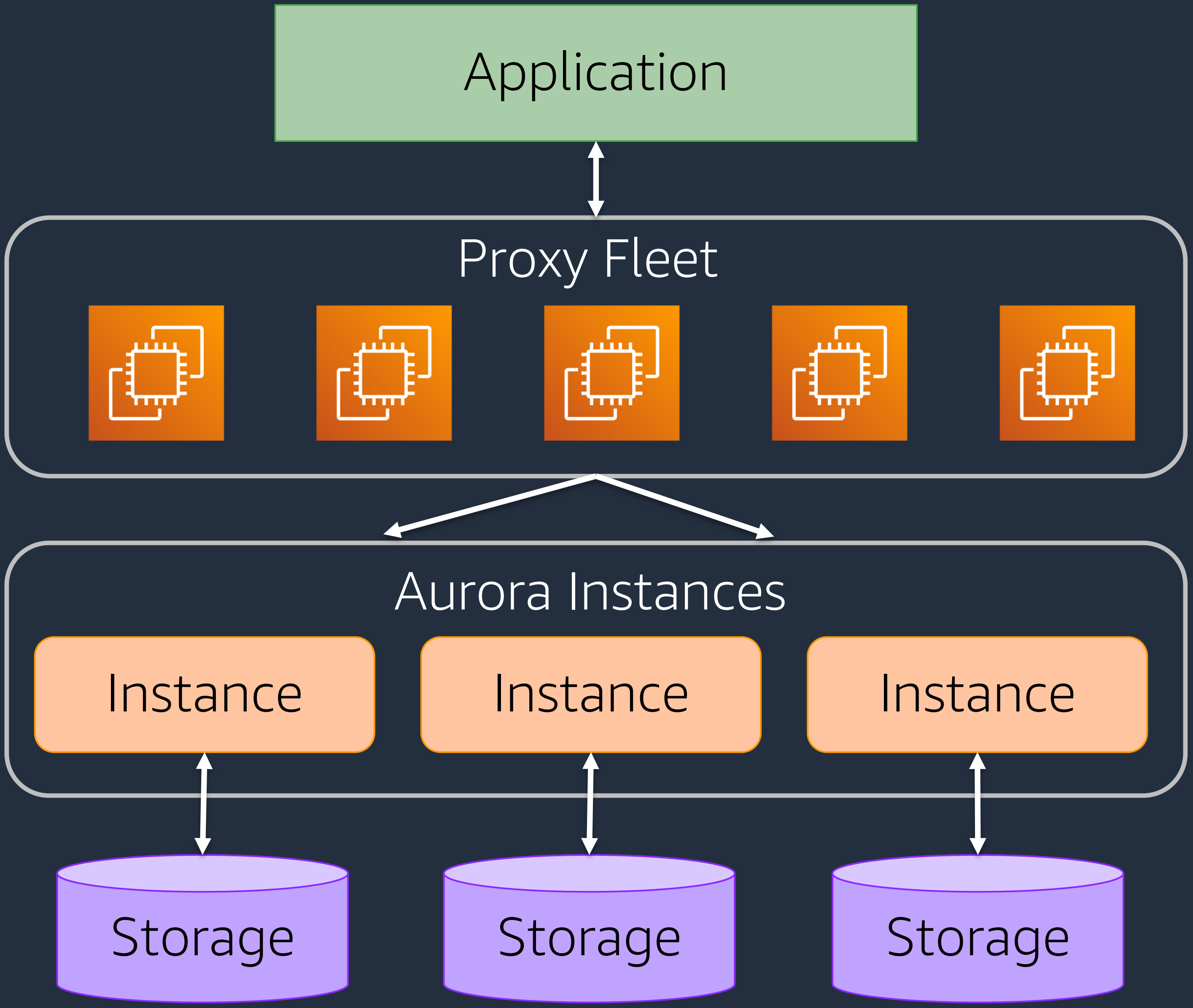
The sizing challenge



Aligning consumption with cost can be challenging

- How do you accommodate different size tenants (noisy neighbor)?
- How do you prevent over-provisioning?
- How do you optimize based on actual consumption?
- How will you support tiering and SLAs?

Serverless storage to the rescue



- Remove the notion of servers/instances
- All data is kept in highly available storage volume
- Application talks to a MySQL or PostgreSQL compatible endpoint
- Fleet of proxy servers manage, queue and route database traffic

Thank You!