SERVERLESS EXPRESS LEAGUE

Building Multi-tenant SaaS Application using Serverless

Anubhav Sharma
Principal Solutions Architect
AWS SaaS Factory
antonal80

Anubhav Sharma
Principal Solutions Architect
AWS SaaS Factory
anubhavynr

SaaS architecture challenges

Tenant isolation

Observability

Noisy Neighbors

Identity Management

Tiering strategy

Cost per tenant

What do you need to build an application

Serverless Compute

Serverless Database

Serverless API Gateway

Serverless Orchestration

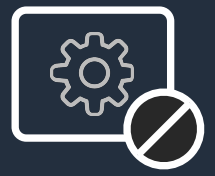
Serverless Messaging

Serverless Event Bus

Serverless Notifications

Serverless Observability

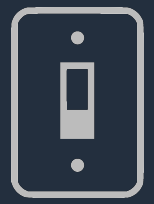
What is Serverless



No infrastructure provisioning, no management

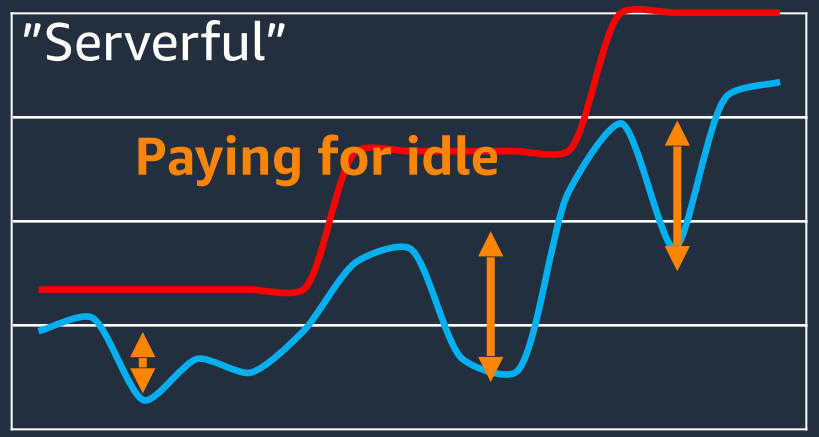
Automatic scaling

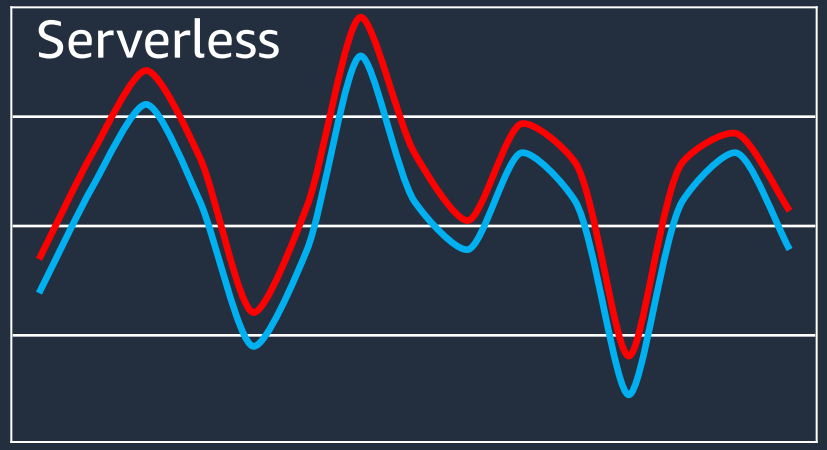
Pay for value



Highly available and secure

Serverless aligns SaaS costs with utilization





Capacity cost Capacity utilization

Agility

Performance

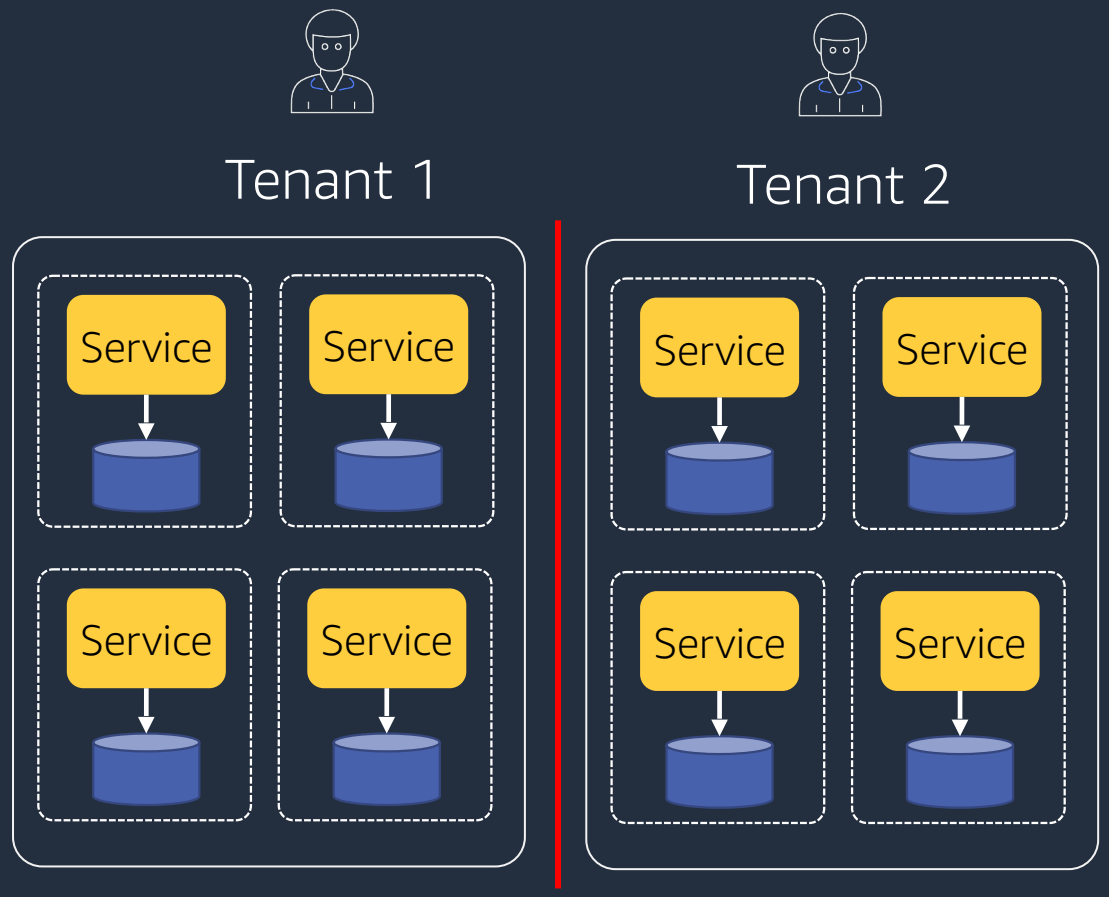
Cost

Security

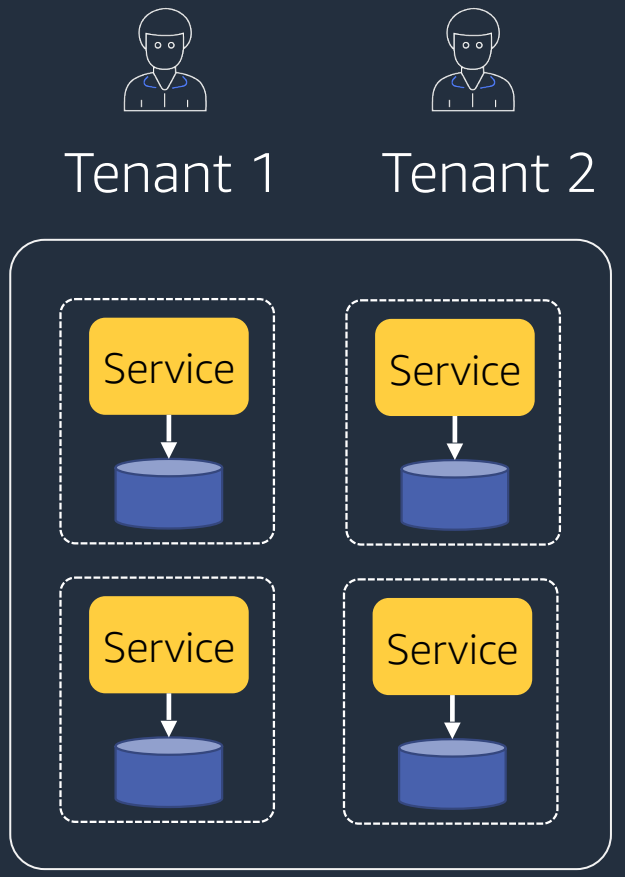
Using Serverless allows you to have the best capacity cost to capacity utilization ratio.

aws

SaaS deployment models



Dedicated resources for each tenant
(silo model)



Shared resources for all tenants
(pool model)

AWS Services & Features Used

API Gateway
Rest API, Lambda Authorizer, Usage Plans, API Keys

Cognito
User pools

Lambda
Fine-grained access control with STS, Layers

DynamoDB
Data persistence

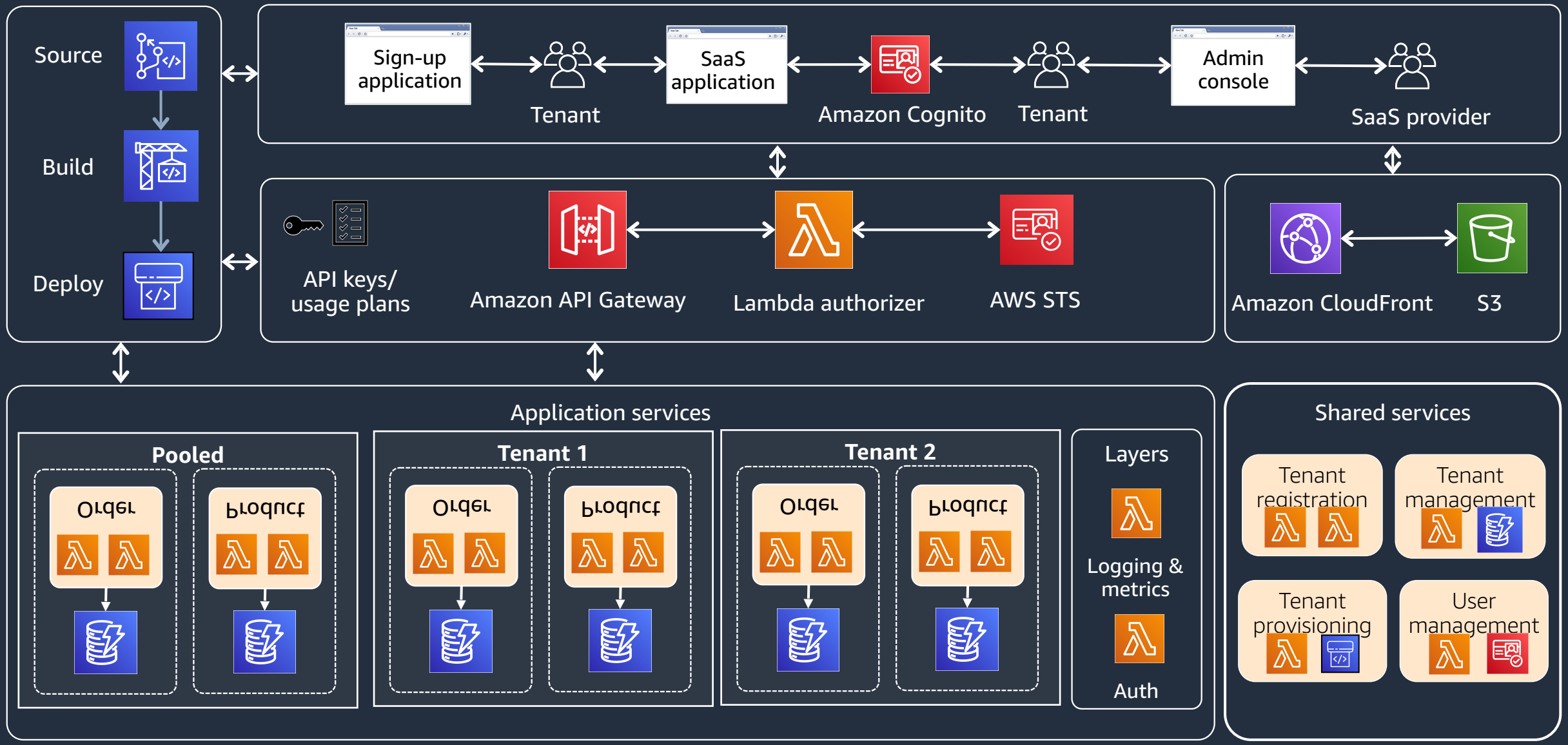
SAM

CDK

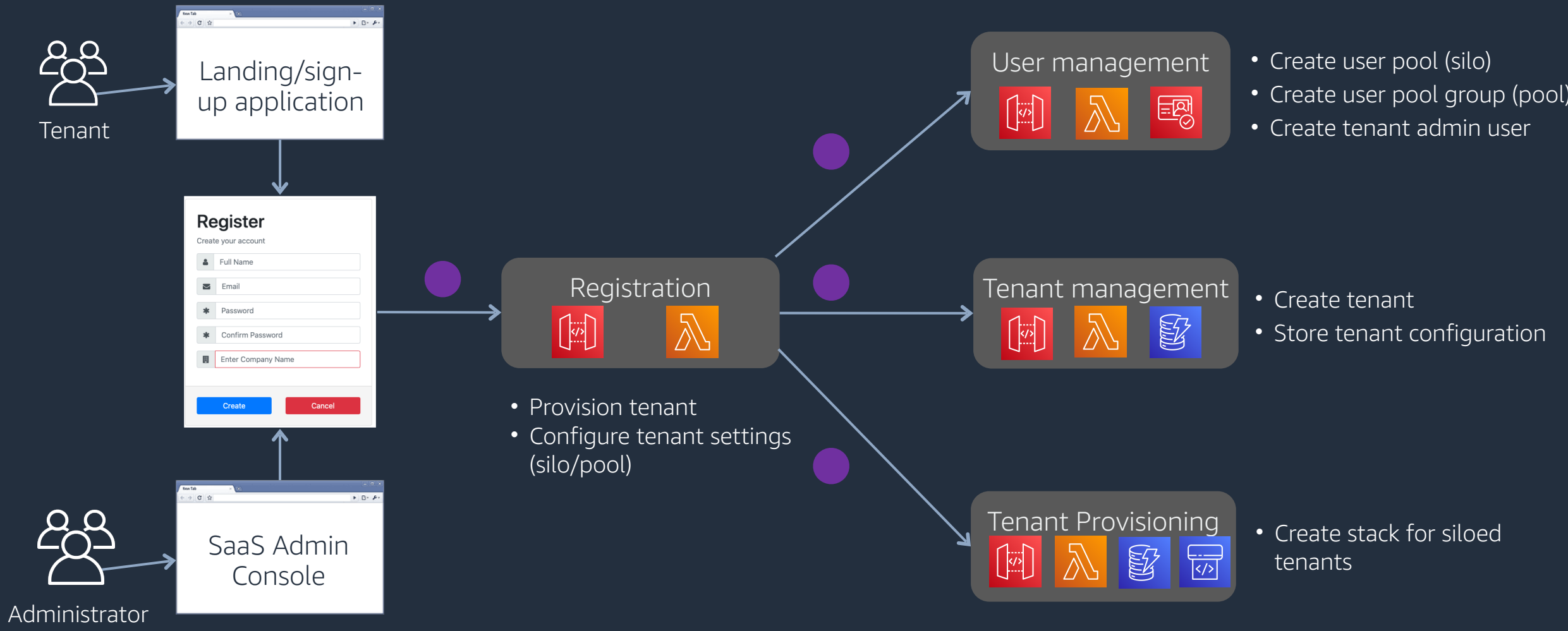
Code Pipeline
Canary deployments

SERVERLESS EXPRESS LEAGUE

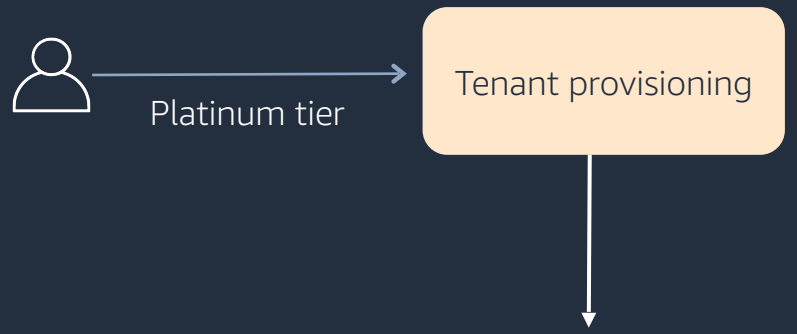
Overall architecture



Registering new tenants

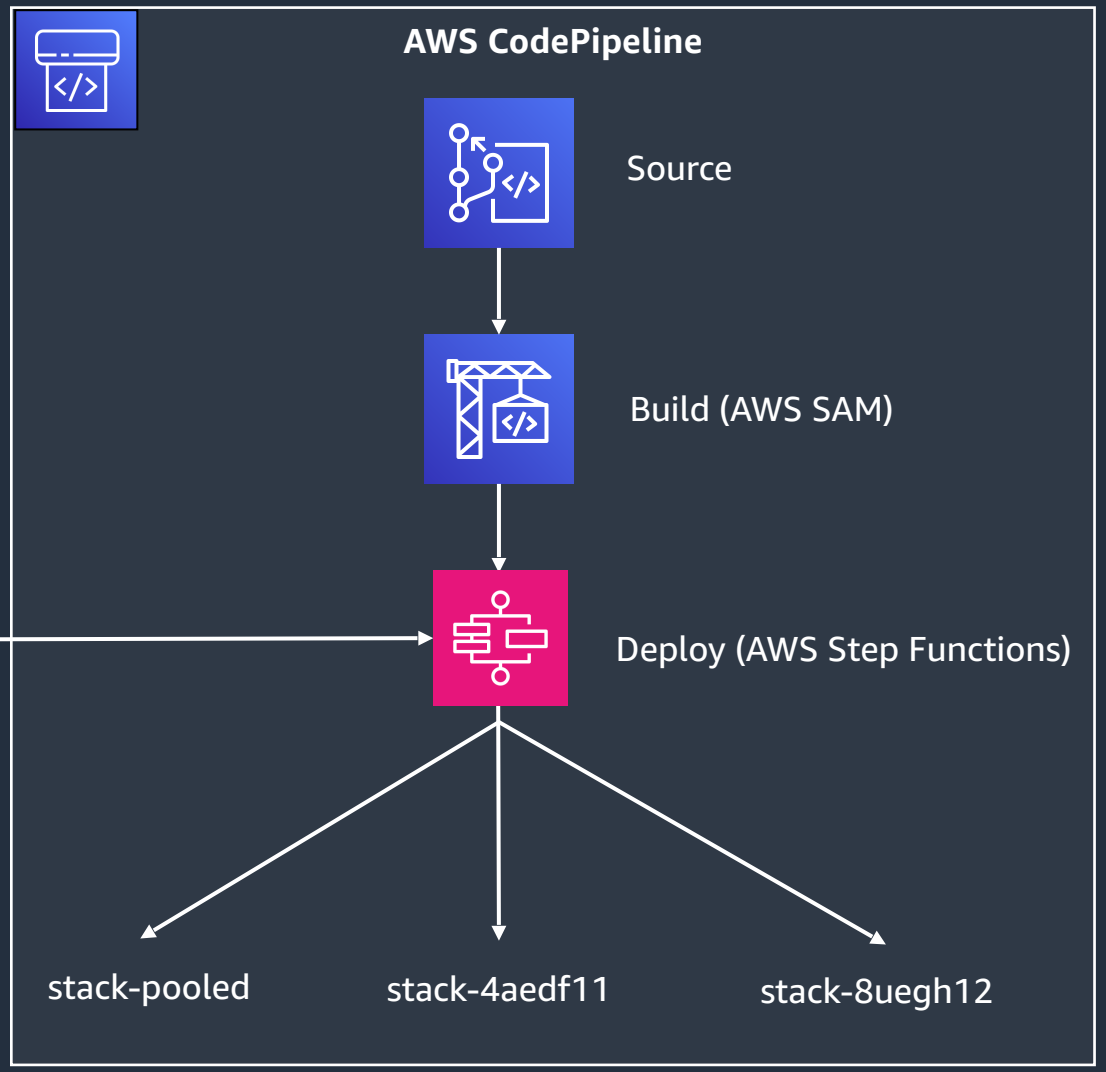


Tenant provisioning and deployment

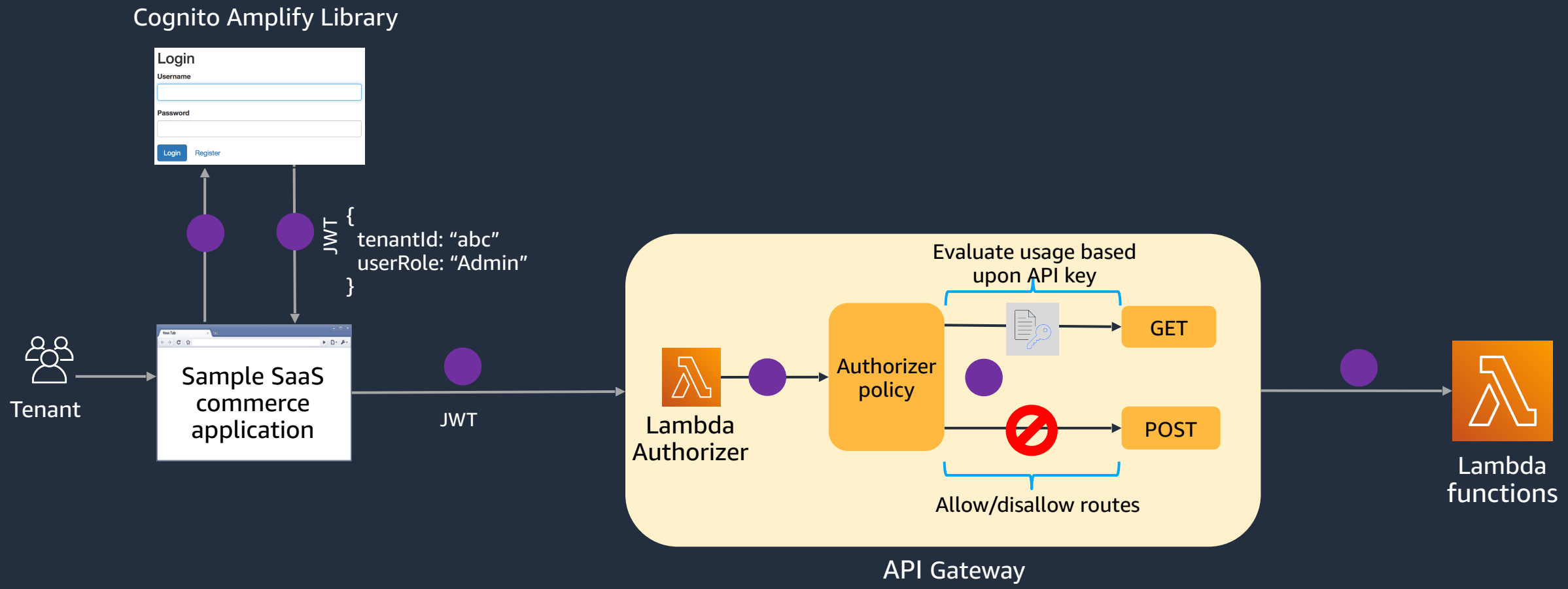


TenantStackMapping

tenantId	waveNumber	codeCommitId	stackName
pooled	1	0a65d	stack-pooled
tenant1	2	0a65d	stack-tenant1
tenant2	2	0a65d	stack-tenant2



Authentication and authorization



Lambda Authorizer policy and context

```
#only tenant admin and system admin can do certain actions like create and disable users
if (auth_manager.isTenantAdmin(user_role) or auth_manager.isSystemAdmin(user_role)):
    policy.allowAllMethods()
    if (auth_manager.isTenantAdmin(user_role)):
        policy.denyMethod(HttpVerb.POST, "tenant-activation")
        policy.denyMethod(HttpVerb.GET, "tenants")
else:
    #if not tenant admin or system admin then only allow to get info and update info
    policy.allowMethod(HttpVerb.GET, "user/*")
    policy.allowMethod(HttpVerb.PUT, "user/*")

authResponse = policy.build()
```

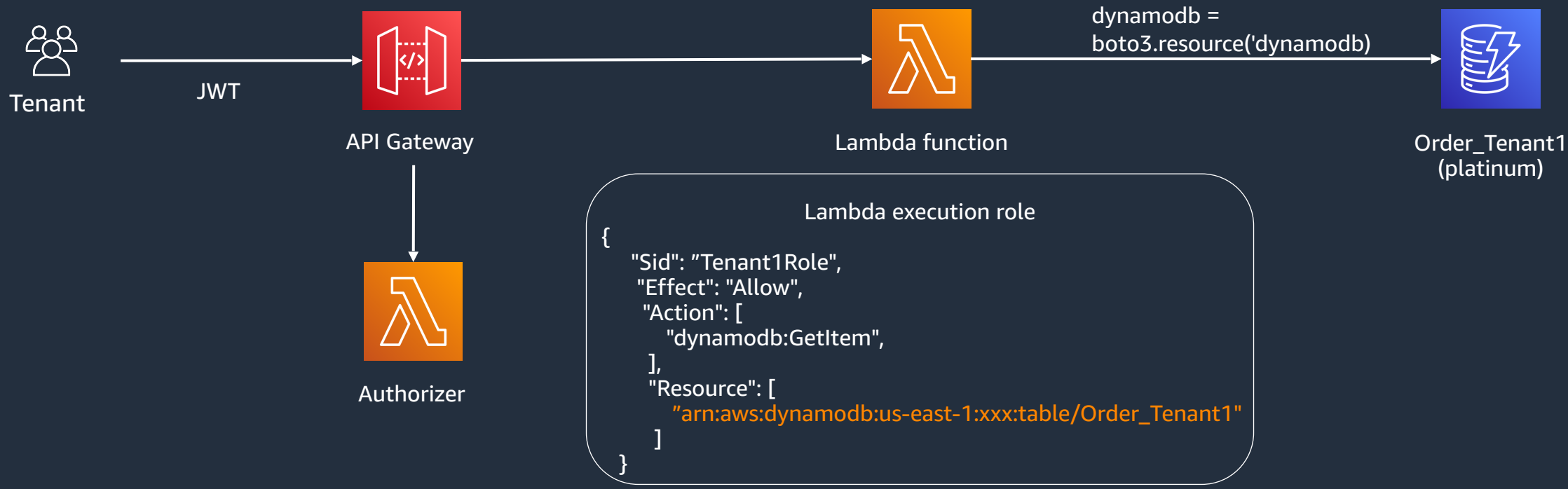
Policy to allow/deny routes based on User Role

```
context = {
    'userName': user_name,
    'tenantId': tenant_id,
    'userPoolId': userpool_id,
    'apiKey': api_key,
    'userRole': user_role
}

authResponse['context'] = context
authResponse['usageIdentifierKey'] = api_key

return authResponse
```

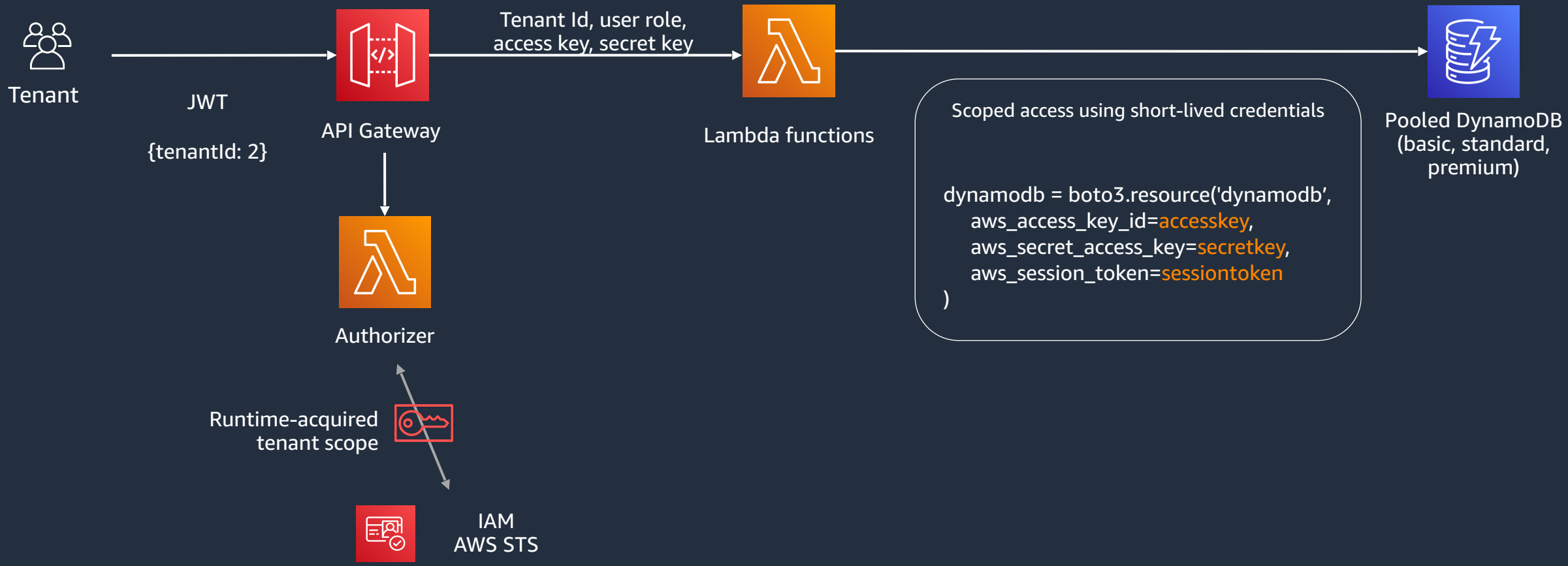
Context with tenant_id and api_key

```
x-amazon-apigateway-api-key-source : "AUTHORIZER"
```

Tenant isolation: Silo model



Tenant isolation: Pooled model



Dynamic policy

```
{
   "Effect": "Allow",
   "Action": [
      "dynamodb:UpdateItem",
      "dynamodb:GetItem",
      "dynamodb:PutItem",
      "dynamodb:DeleteItem",
      "dynamodb:Query"
   ],
   "Resource": [
      "arn:aws:dynamodb:{0}:{1}:table/Product-*".format(region, aws_account_id),
   ],




   }
}
```

Code snippet: STS credentials

```
#get IAM policy
iam_policy = getPolicyForUser(user_role, tenant_id)

#use STS client to generate the credentials
assumed_role = sts_client.assume_role(
                RoleArn=role_arn,
                RoleSessionName="tenant-aware-session",
                Policy=iam_policy,
)
credentials = assumed_role["Credentials"]

#pass sts credentials to lambda
context = {
                'accesskey': credentials['AccessKeyId'], # $context.authorizer.key -> value
                'secretkey' : credentials['SecretAccessKey’],
                'sessiontoken' : credentials["SessionToken"],
}
```

Tier applied in data access layer

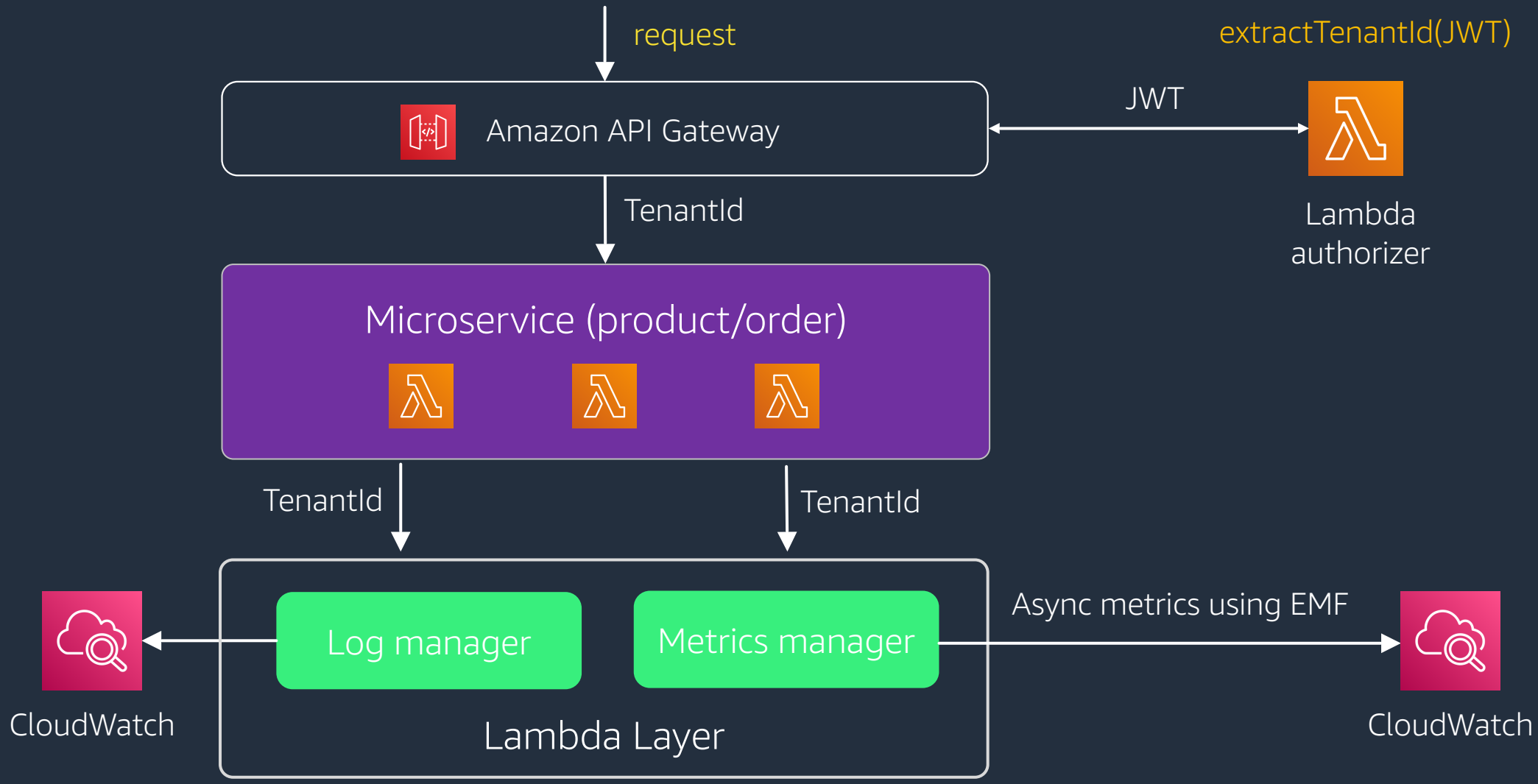
```
142    def __get_dynamodb_table(event, dynamodb):
143        if (is_pooled_deploy=='true'):
144            accesskey = event['requestContext']['authorizer']['accesskey']
145            secretkey = event['requestContext']['authorizer']['secretkey']
146            sessiontoken = event['requestContext']['authorizer']['sessiontoken']
147            dynamodb = boto3.resource('dynamodb',
148                    aws_access_key_id=accesskey,
149                    aws_secret_access_key=secretkey,
150                    aws_session_token=sessiontoken
151                    )
152        else:
153            if not dynamodb:
154                dynamodb = boto3.resource('dynamodb')
155
156        return dynamodb.Table(table_name)
157
```

DynamoDB partition key structure

ShardID	ProductID	doc
tenant1-1	1	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 1", "SKU": 750, "ProductPrice": 2709}
tenant1-4	2	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 2", "SKU": 750, "ProductPrice": 1700}
tenant1-7	3	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 3", "SKU": 41, "ProductPrice": 1885}
tenant1-9	8	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 9", "SKU": 46, "ProductPrice": 1540}
tenant2-1	4	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 4", "SKU": 766, "ProductPrice": 1081}
tenant2-4	6	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 6", "SKU": 692, "ProductPrice": 677}
tenant2-5	5	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 5", "SKU": 692, "ProductPrice": 677}
tenant2-6	7	{"Description": "PRODUCT DESCRIPTION FOR PRODUCT ID : 7", "SKU": 577, "ProductPrice": 3211}

https://aws.amazon.com/blogs/apn/partitioning-pooled-multi-tenant-saas-data-with-amazon-dynamodb/

Abstract Tenant Details with Lambda Layers



SERVERLESS EXPRESS LEAGUE

Observability

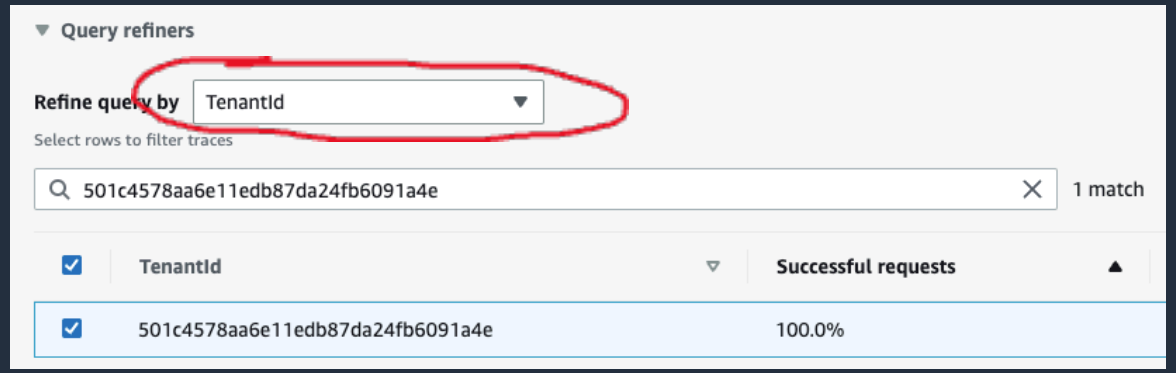
Logging with tenant context

```
2021-09-21T18:26:55.488-04:00    EXTENSION Name: cloudwatch_lambda_agent
2021-09-21T18:26:55.518-04:00    {"level":"INFO","location":"log_with_ten

{
    "level": "INFO",
    "location": "log_with_tenant_context:19",
    "message": "Request received to create a product",
    "timestamp": "2021-09-21 22:26:55,499+0000",
    "service": "ProductService",
    "tenant_id": "4fb565351b0a11ecab15250a6772c4a2",
    "xray_trace_id": "1-614a5c2e-76c4853c4e86383723b27a9d"
}
```

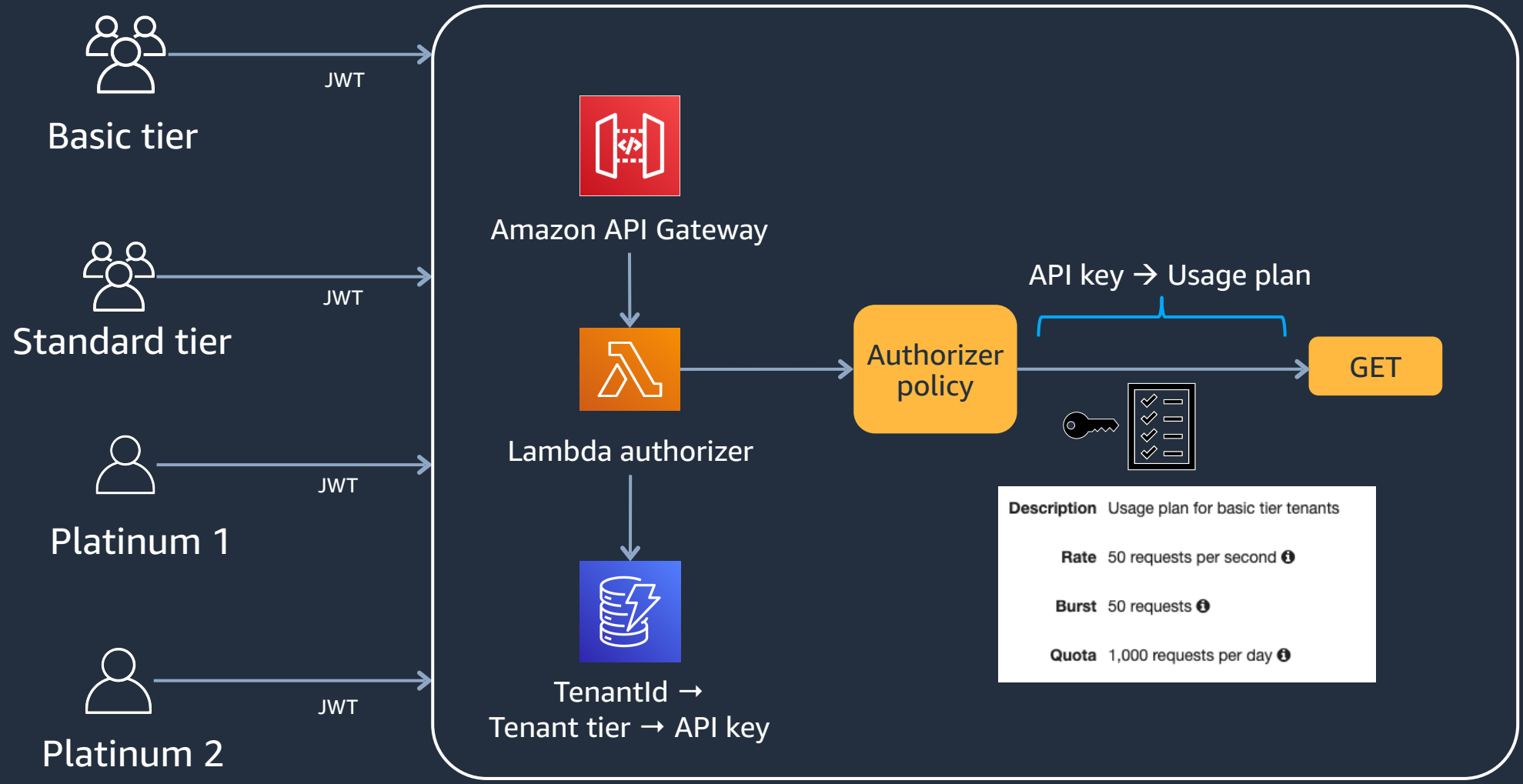
Multi-tenant metrics (EMF)

```
2021-08-24T17:39:07.102-04:00    {"_aws": {"Timestamp": 1629841147102, "CloudWatchMetrics": [{"Nam

{
    "_aws": {
        "Timestamp": 1629841147102,
        "CloudWatchMetrics": [
            {
                "Namespace": "ServerlessSaaS",
                "Dimensions": [
                    [
                        "tenant_id",
                        "service"
                    ]
                ],
                "Metrics": [
                    {
                        "Name": "ProductCreated",
                        "Unit": "Count"
                    }
                ]
            }
        ]
    },
    "tenant_id": "fc651ed9052211eca250e538bb53ea77",
    "service": "ProductService",
    "ProductCreated": [
        1
    ]
}
```

Multi-tenant tracing with annotations



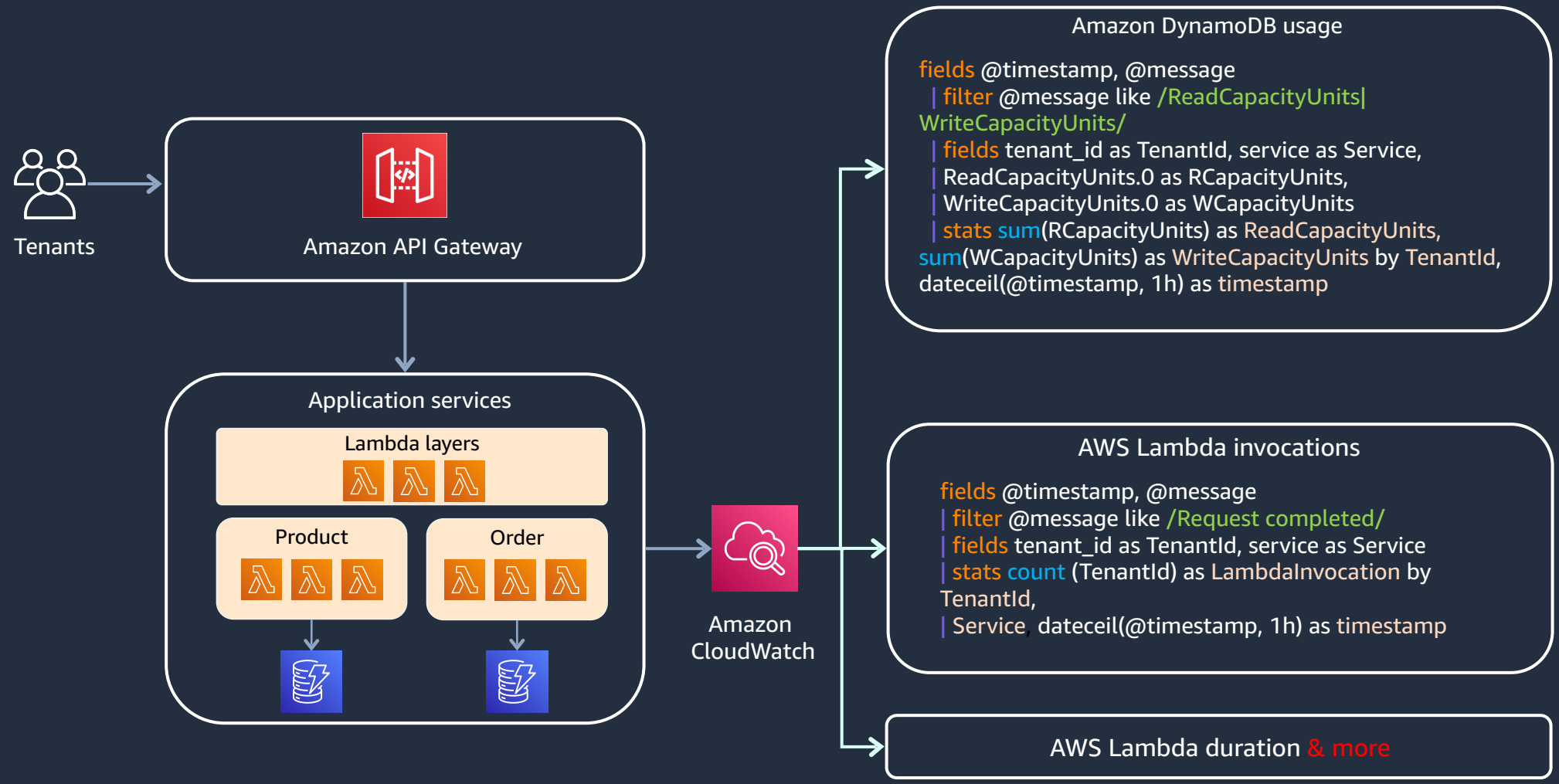
Tier-based throttling policies



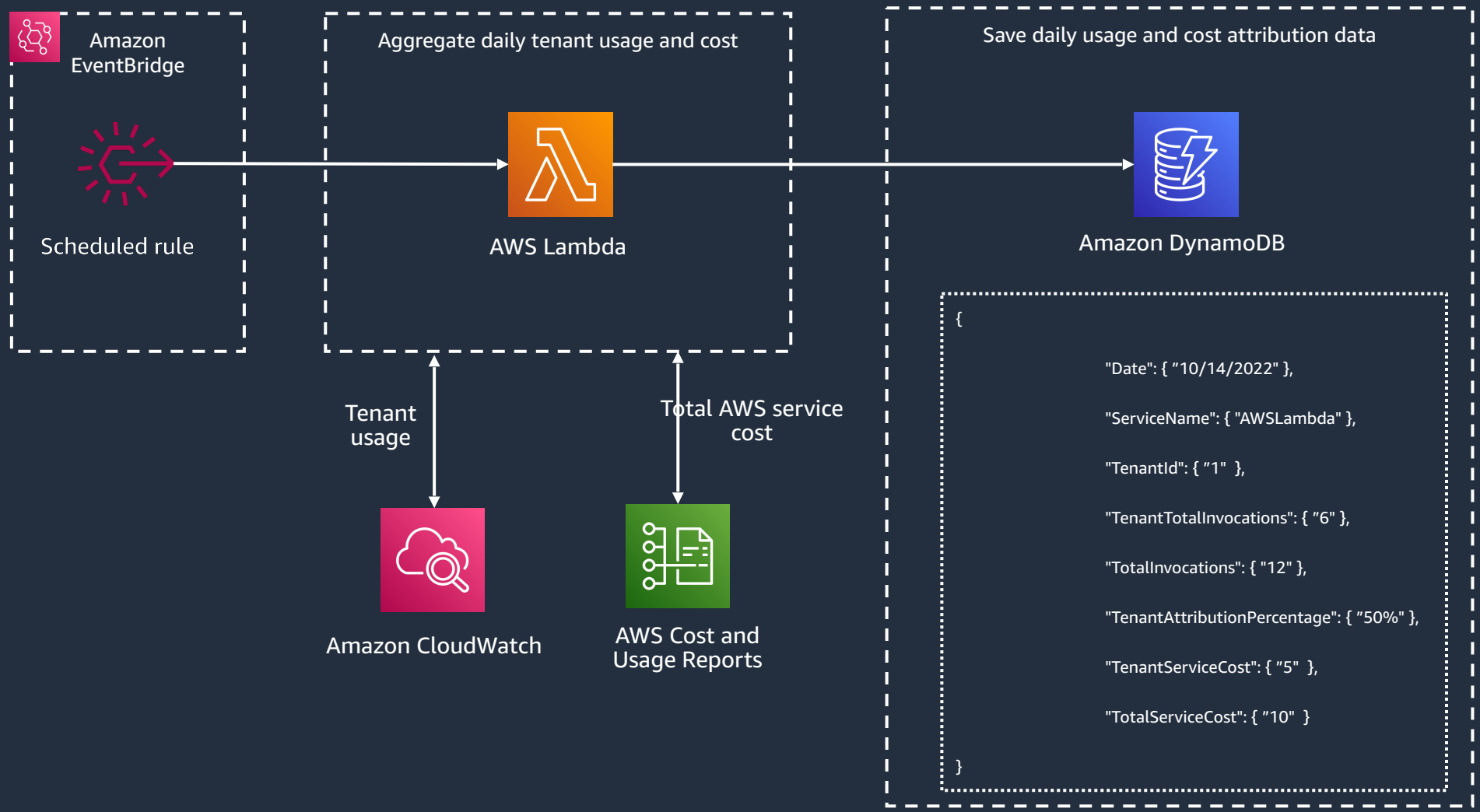
Cost attribution in a pooled model

- Or do you just need consumption?
- Capturing tenant level metrics is the key
- It will be always an approximation
- Capture metrics that are relevant

Cost attribution in a pooled model



Cost attribution in a pooled model



Serverless SaaS reference links

Building SaaS with Serverless blog

Serverless SaaS Workshop

Serverless SaaS Workshop (GitHub Repo)

Serverless SaaS Reference Solution

- https://aws.amazon.com/blogs/apn/building-a-multi-tenant-saas-solution-using-aws-serverless-services/
- https://catalog.us-east-1.prod.workshops.aws/workshops/b0c6ad36-0a4b-45d8-856b-8a64f0ac76bb/en-US
- https://github.com/aws-samples/aws-serverless-saas-workshop
- https://github.com/aws-samples/aws-saas-factory-ref-solution-serverless-saas

Meet us at re:Invent

- SAS404 | Designing and implementing a SaaS cost-per-tenant strategy
- SAS406 | SaaS DevOps deep dive: Automating multi-tenant deployments
- SAS403 | SaaS survivor: Building a rich multi-tenant operations experience
- SVS320 | Building multi-tenant applications with AWS Lambda and AWS Fargate
- API306 | Combining Step Functions and EventBridge: Use cases and best practices

Continue your AWS serverless learning

Learn at your own pace

Expand your serverless skills with our learning plan on **AWS Skill Builder**

Increase your knowledge

Use our **Ramp-Up Guides** to build your serverless knowledge

Earn AWS Serverless badge

Demonstrate your knowledge by achieving **digital badges**

https://s12d.com/serverless-learning

Thank you

Anubhav Sharma
Principal Solutions Architect
AWS SaaS Factory
antonal80

Anubhav Sharma
Principal Solutions Architect
AWS SaaS Factory
anubhavynr