Building a SaaS solution on AWS using Serverless

Anubhav Sharma

Principal Solutions Architect
AWS SaaS Factory

Software-as-a-Service (SaaS) is a business and software delivery model that enables organizations to offer their solution in a low-friction, service-centric approach.

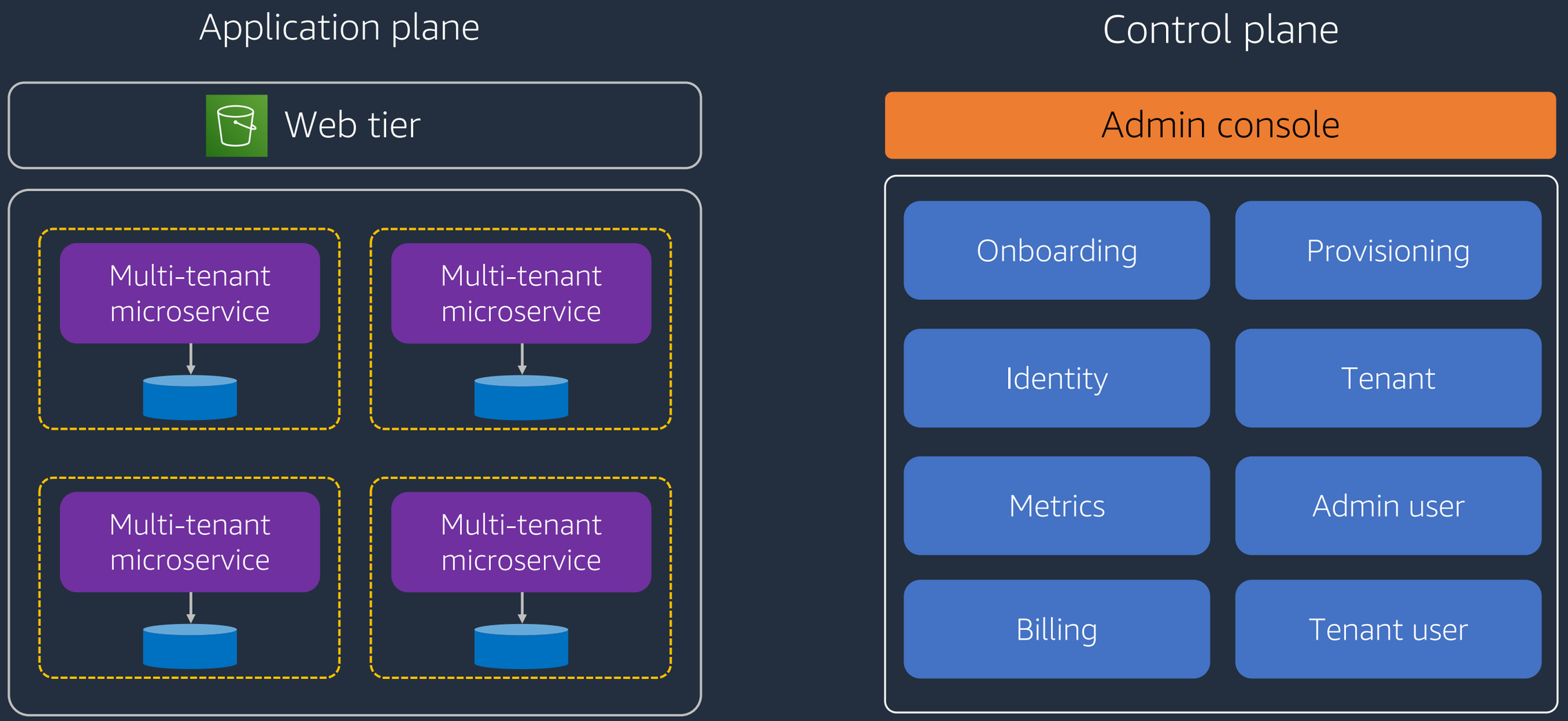
> “There are many advantages to a customer-centric approach, but here’s the big one: Customers are always beautifully, wonderfully dissatisfied, even when they report being happy and business is great. Even when they don’t yet know it, customers want something better, and your desire to delight customers will drive you to invent on their behalf.”

Jeff Bezos

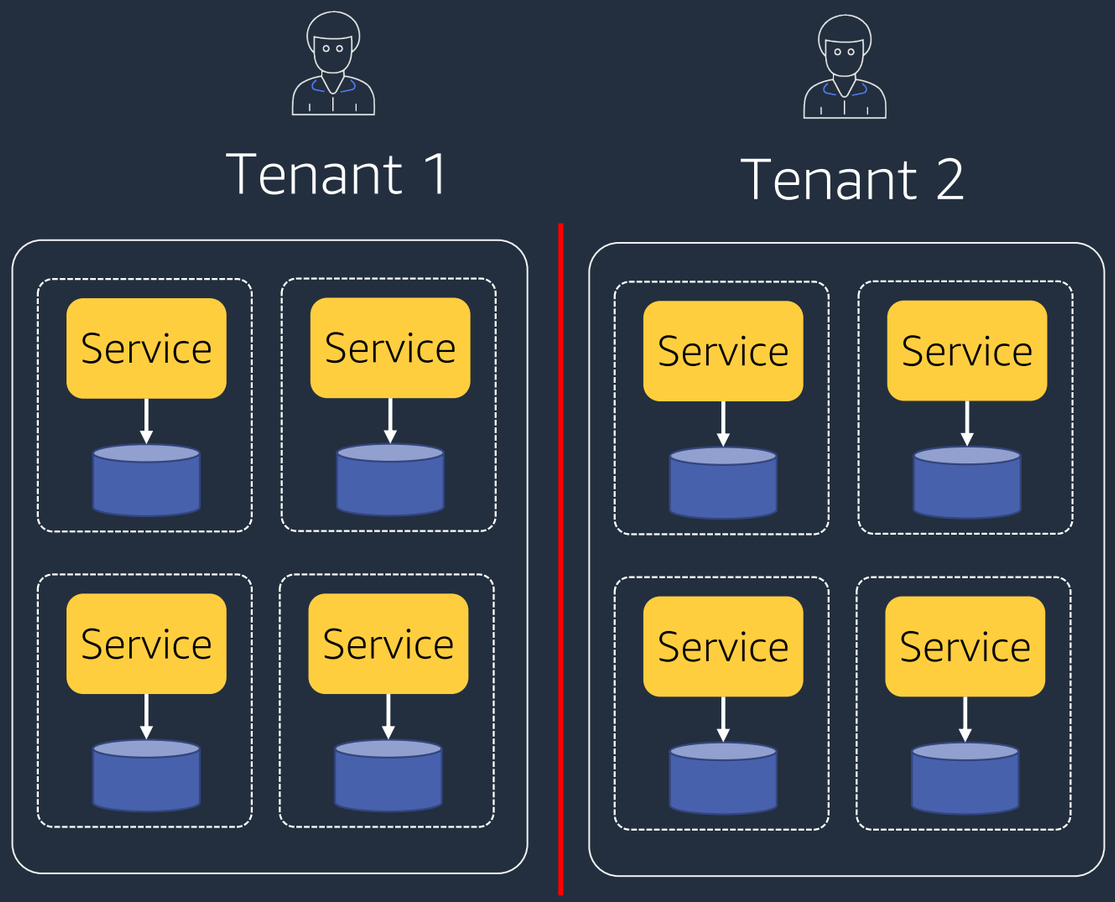
Founder and Executive Chair, Amazon.com, Inc.

2016 letter to shareholders

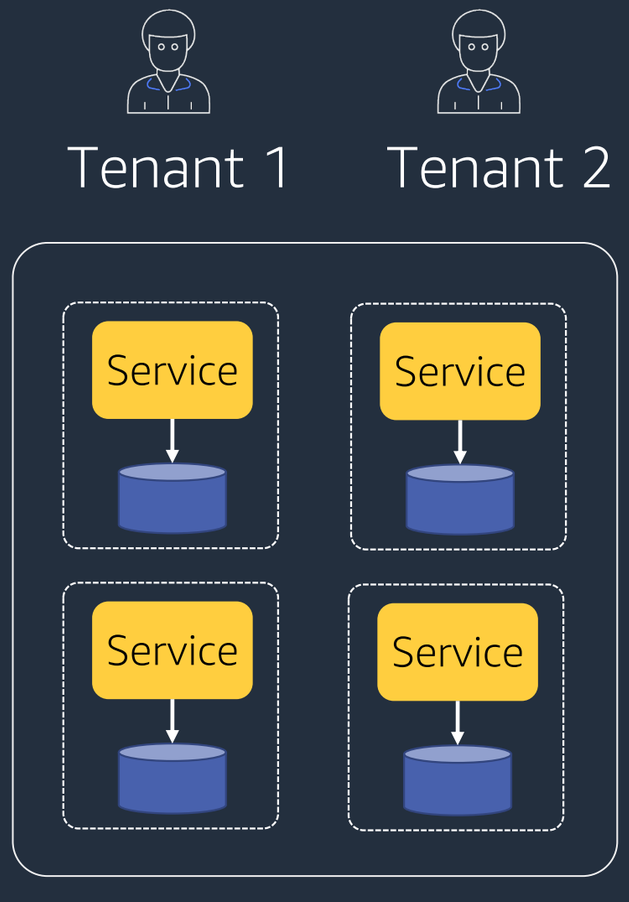
What it means to be SaaS



Application plane deployment models



Dedicated resources for each tenant
(silo model)



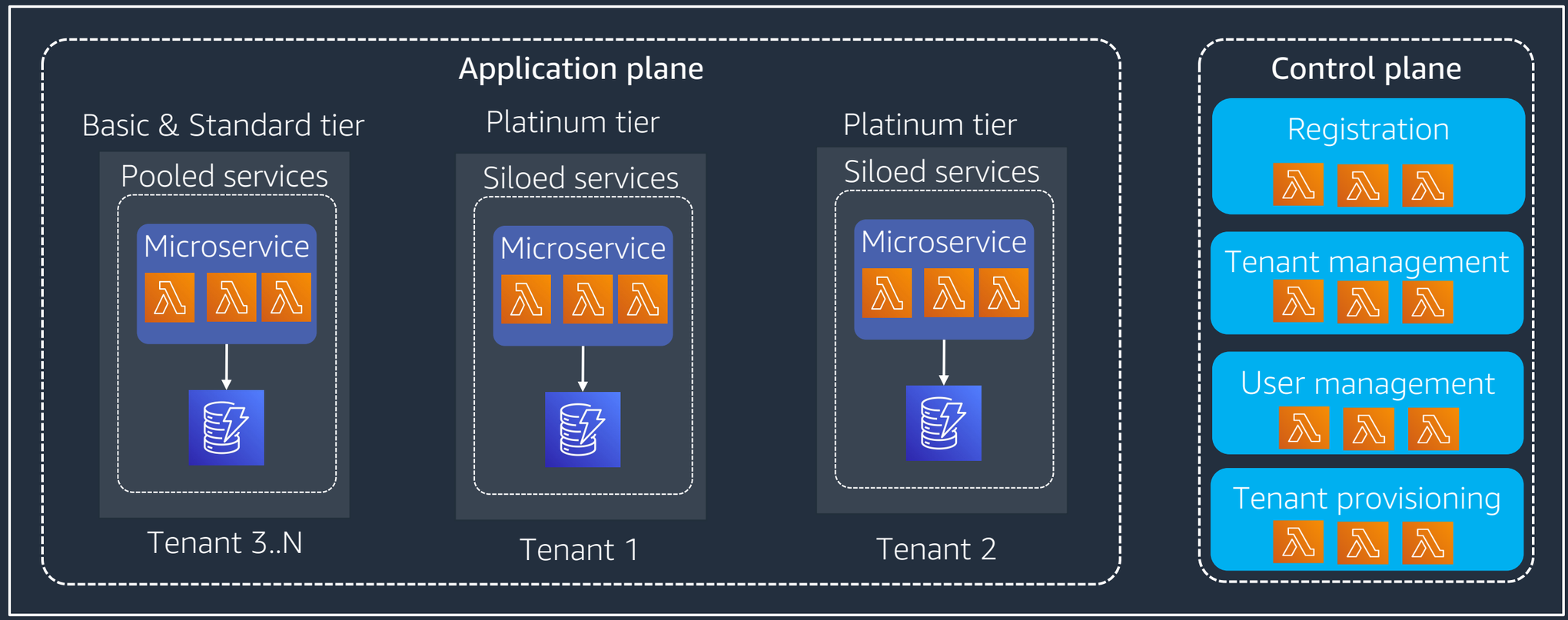
Shared resources for all tenants
(pool model)

Serverless SaaS reference architecture

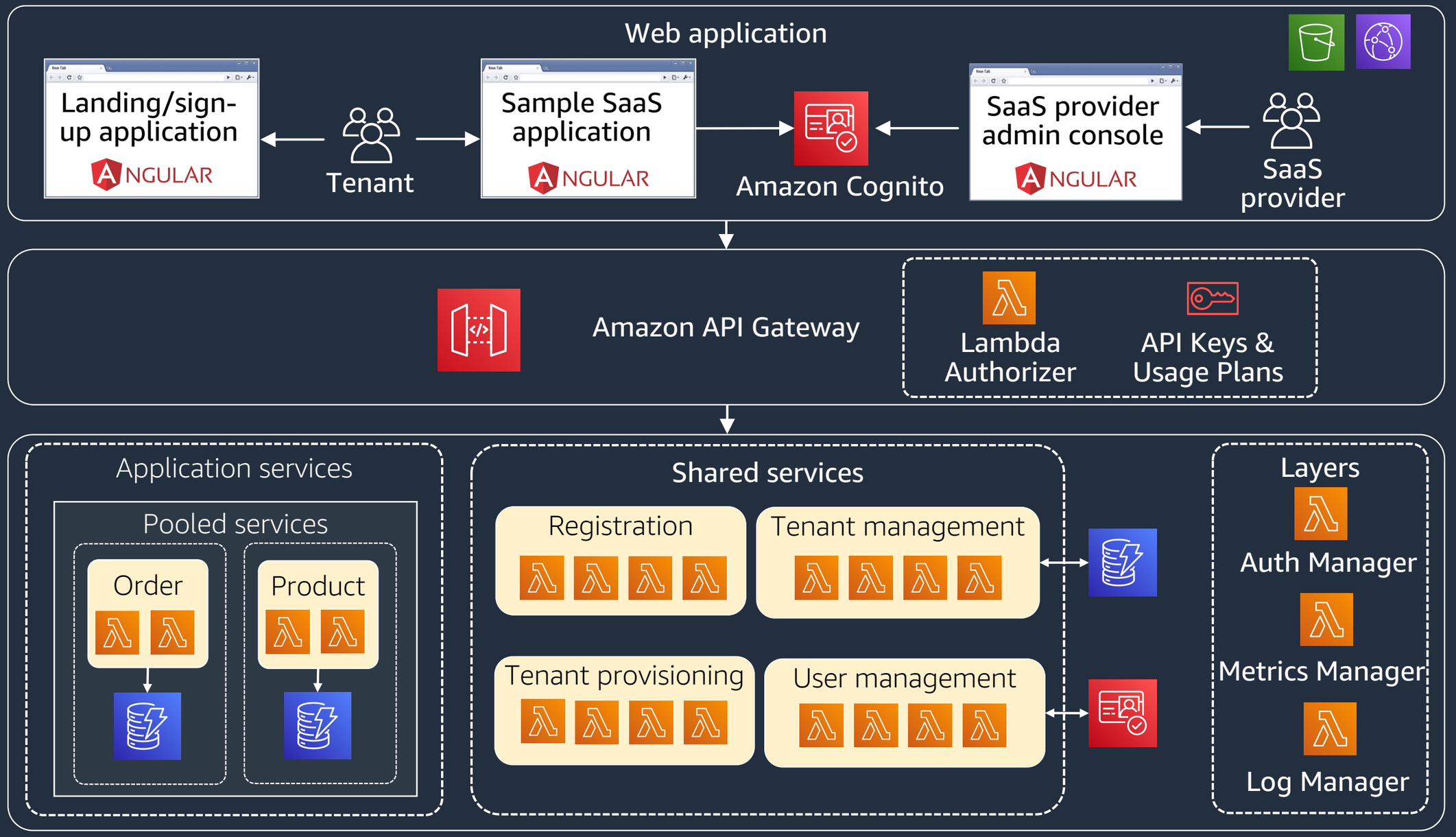
AWS Services & Features Used

- AWS Serverless Application Model (SAM)
- AWS Cloud Development Kit (CDK)
- Amazon API Gateway
    - REST APIs
    - Lambda Authorizer
    - Usage plans & API keys
- Amazon Cognito
    - User Pools
- AWS Lambda
    - Fine-grained access control (AWS STS)
    - Lambda Layers
- Code Pipeline
    - Canary deployments
- Amazon DynamoDB

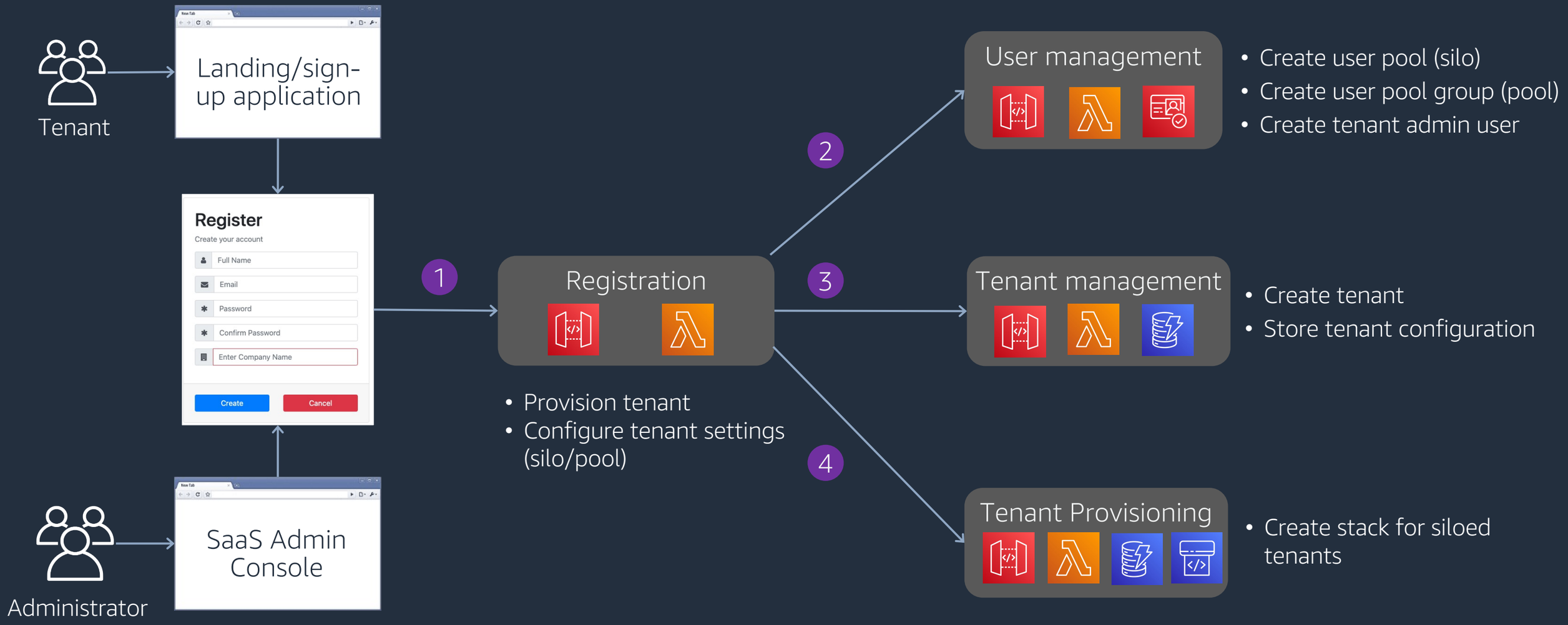
Tiered based deployment model



Baseline Architecture



Registering new tenants



Tenant registration

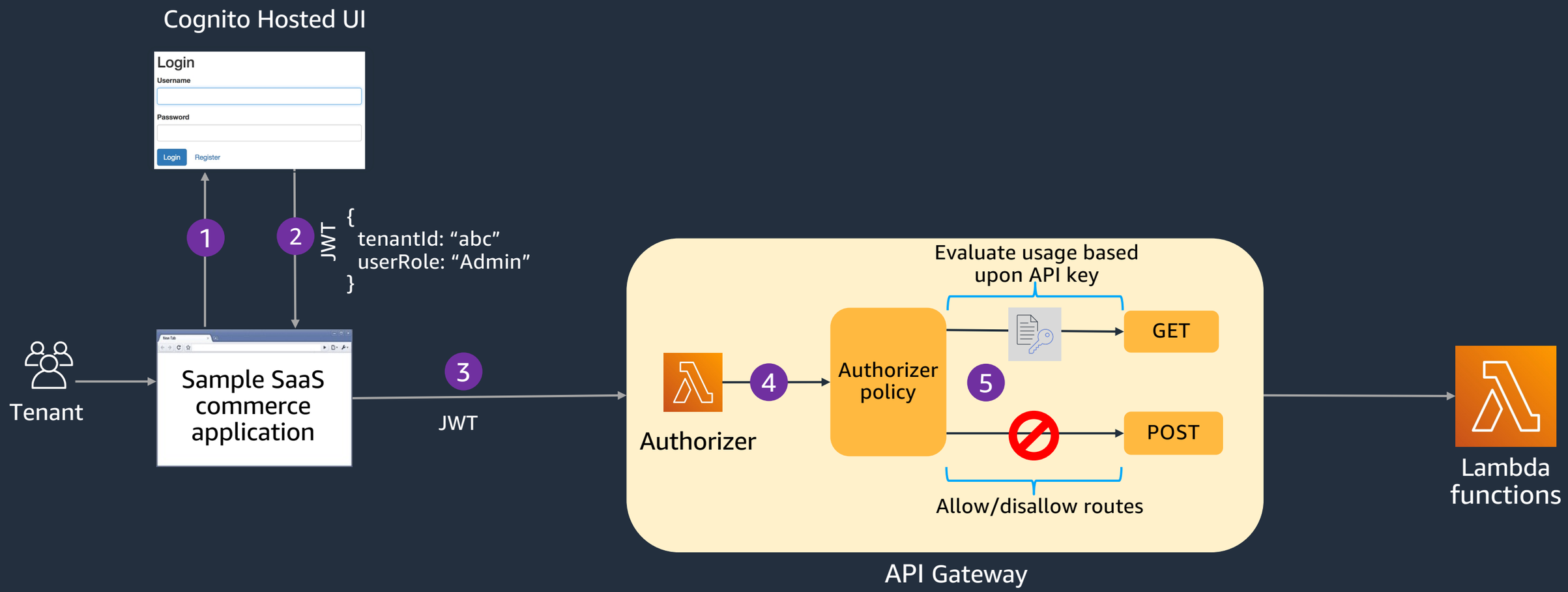
```python
def create_user(event, context):
    user_details = json.loads(event['body'])

    logger.info("Request received to create new user")
    logger.info(event)

    tenant_id = user_details['tenantId']

    response = client.admin_create_user(
        Username=user_details['userName'],
        UserPoolId=user_pool_id,
        ForceAliasCreation=True,
        UserAttributes=[
            {
                'Name': 'email',
                'Value': user_details['userEmail']
            },
            {
                'Name': 'custom:userRole',
                'Value': user_details['userRole']
            },
            {
                'Name': 'custom:tenantId',
                'Value': tenant_id
            }
        ]
```

Create user

```python
def create_tenant(event, context):

    try:
        if(tenant_details['dedicatedTenancy'].lower()!= 'true'):
            settings_response = table_system_settings.get_item(
                Key={
                    'settingName': 'apiGatewayUrl-Pooled'
                }
            )
            api_gateway_url = settings_response['Item']['settingValue']

        response = table_tenant_details.put_item(
            Item={
                    'tenantId': tenant_details['tenantId'],
                    'tenantName' : tenant_details['tenantName'],
                    'tenantAddress': tenant_details['tenantAddress'],
                    'tenantEmail': tenant_details['tenantEmail'],
                    'tenantPhone': tenant_details['tenantPhone'],
                    'tenantTier': tenant_details['tenantTier'],
                    'apiKey': tenant_details['apiKey'],
                    'userPoolId': tenant_details['userPoolId'],
                    'appClientId': tenant_details['appClientId'],
                    'dedicatedTenancy': tenant_details['dedicatedTenancy'],
                    'isActive': True,
                    'apiGatewayUrl': api_gateway_url
                }
            )
```

Create tenant

Authentication and authorization



Routes based on user roles

```
#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()
```

Tenant isolation: Silo 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),
   ],
   "Condition": {
      "ForAllValues:StringLike": {
         "dynamodb:LeadingKeys": [
            "{0}-*".format(tenant_id)
         ]
      }
   }
}
```

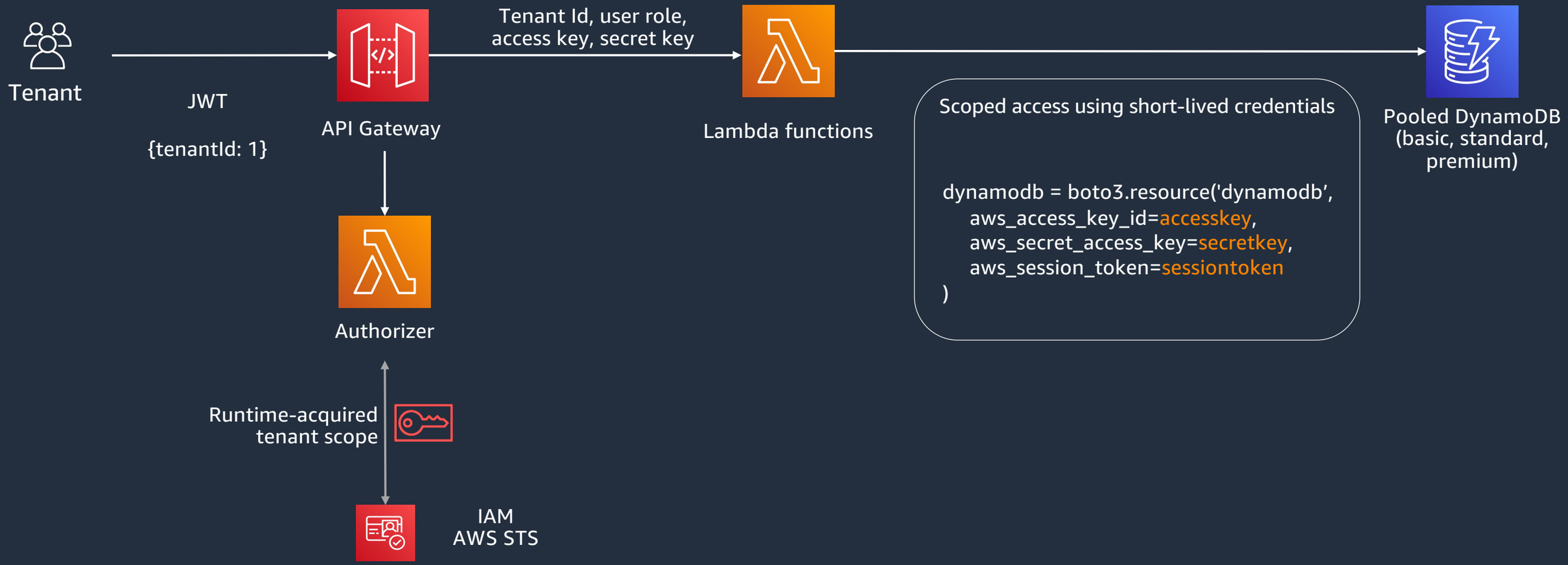
Code snippet: STS credentials

```
#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"],
}
```

Authorizer output

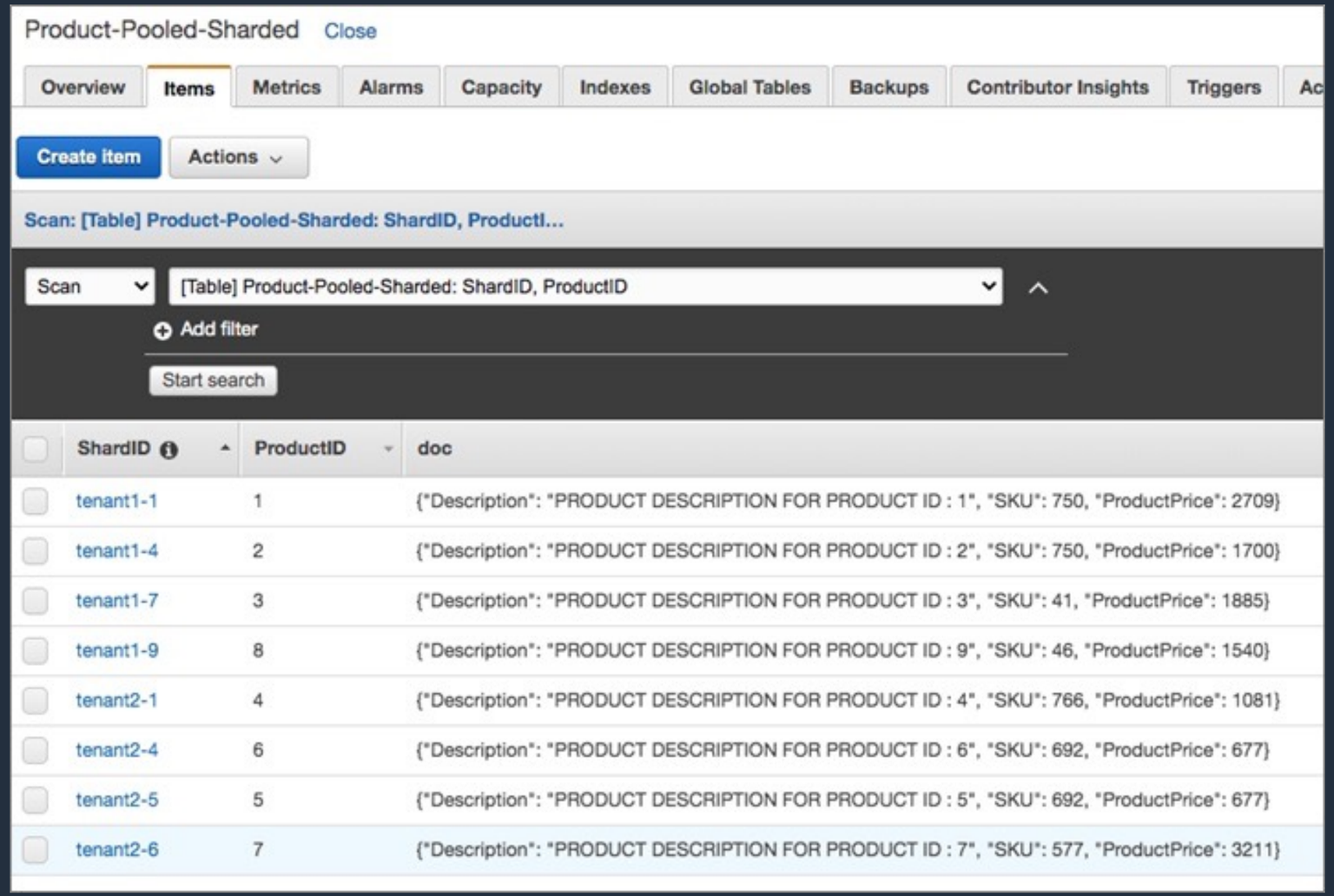
```
#pass sts credentials to lambda
context = {
    'accesskey': credentials['AccessKeyId'], # $context.authorizer.key -> value
    'secretkey' : credentials['SecretAccessKey'],
    'sessiontoken' : credentials["SessionToken"],
    'userName': user_name,
    'tenantId': tenant_id,
    'userPoolId': userpool_id,
    'apiKey': api_key,
    'userRole': user_role
}

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

return authResponse
```

Tenant isolation: Pooled model

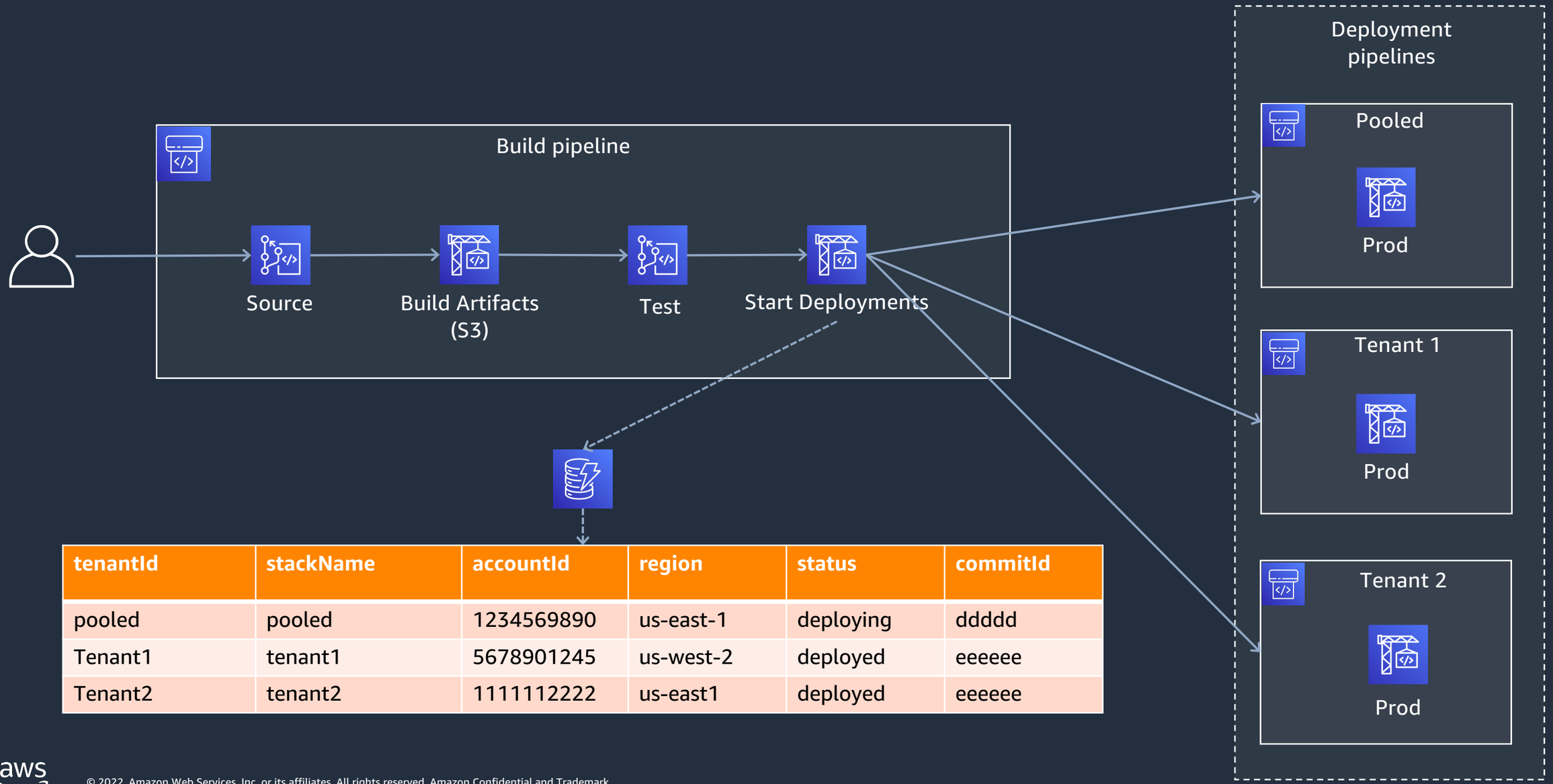


Pool-based partition with DynamoDB

- Tenant data in the same table
- TenantID used as ShardID
- Isolation policies applied by ShardID

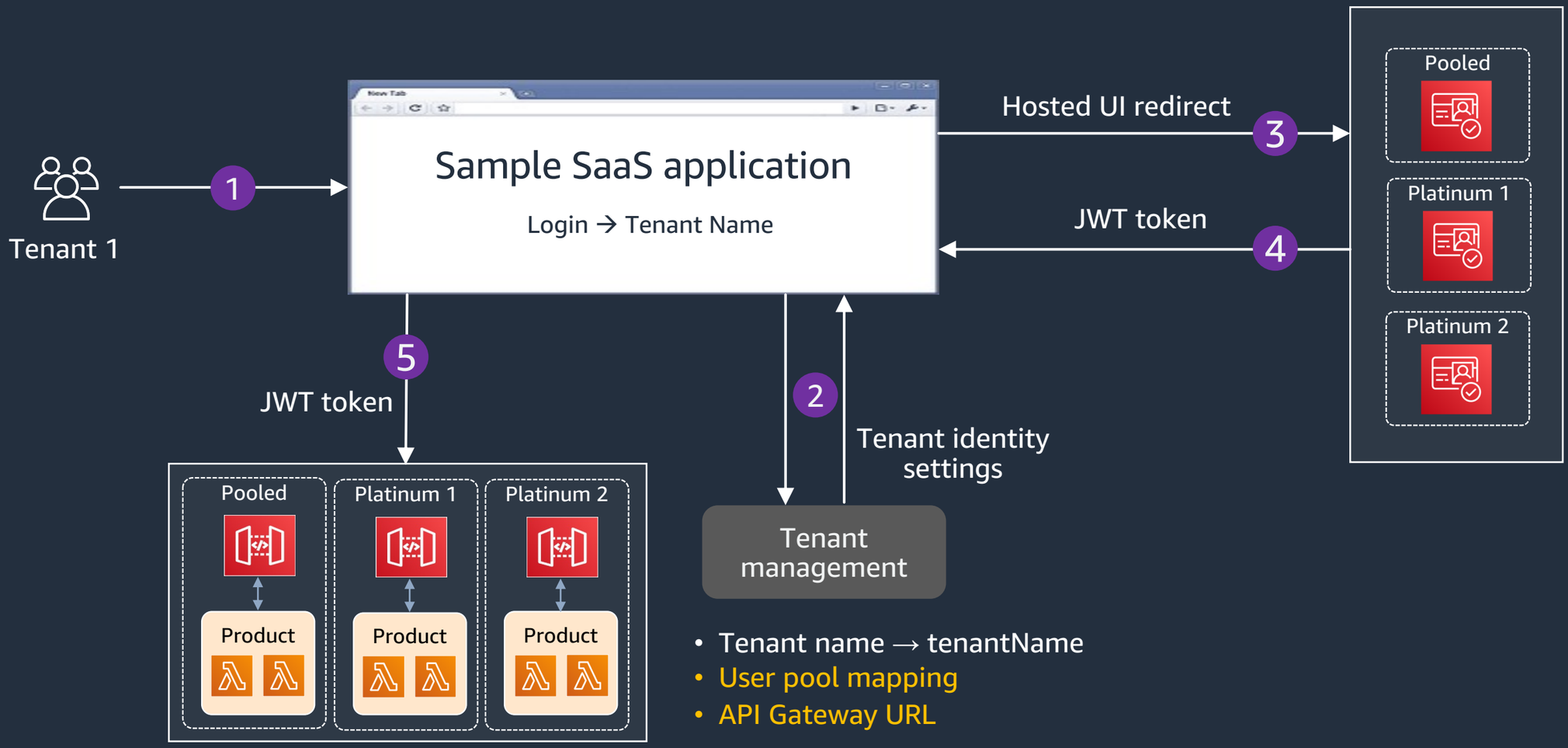


Multi-tier, multi-account, multi-region CI/CD



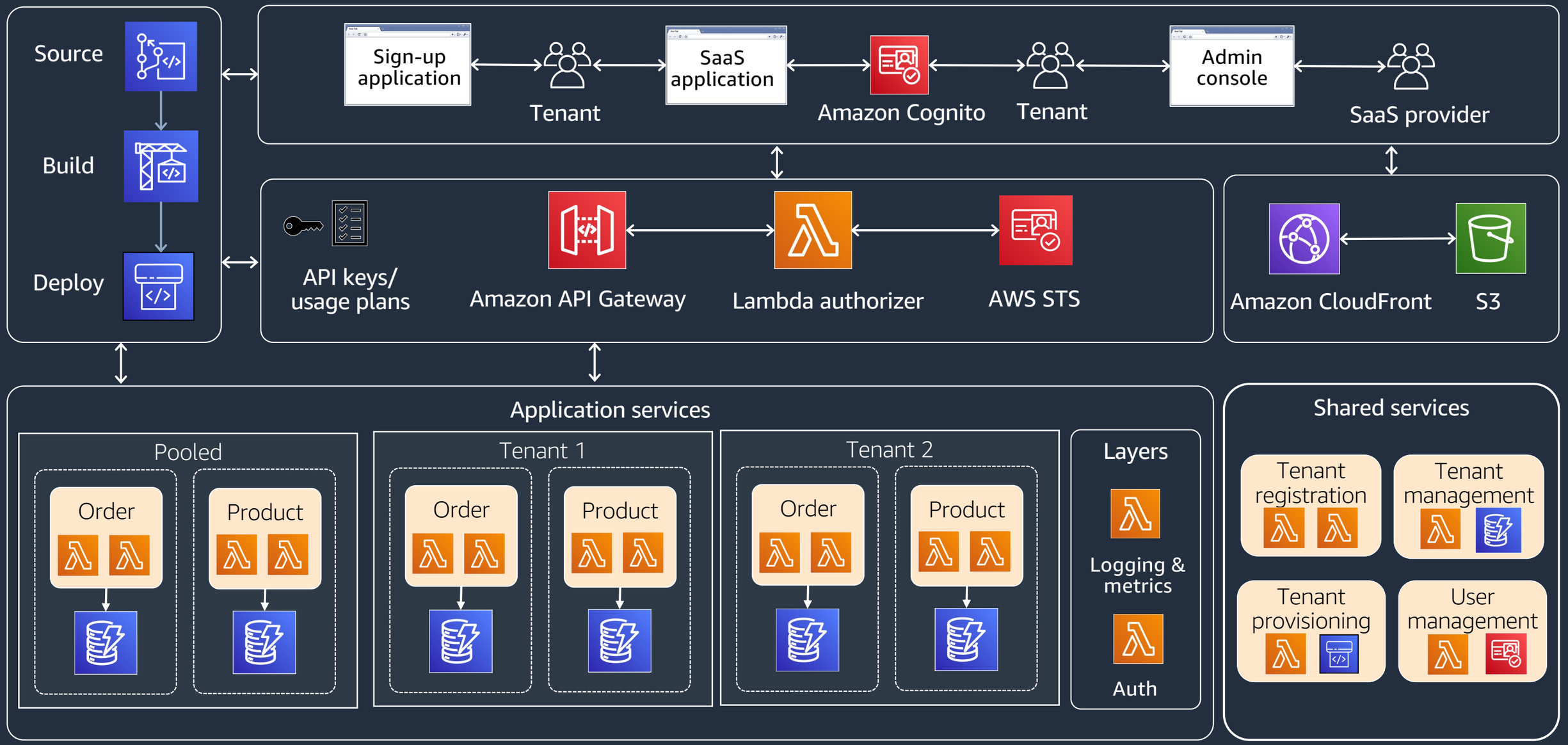
tenantId	stackName	accountId	region	status	commitId
pooled	pooled	1234569890	us-east-1	deploying	ddddd
Tenant1	tenant1	5678901245	us-west-2	deployed	eeeeee
Tenant2	tenant2	1111112222	us-east1	deployed	eeeeee

Tenant routing



- Tenant name → tenantName
- User pool mapping
- API Gateway URL

Final View of Architecture



Serverless SaaS reference links

- 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://aws.amazon.com/blogs/apn/building-a-multi-tenant-saas-solution-using-aws-serverless-services/
- https://github.com/aws-samples/aws-saas-factory-ref-solution-serverless-saas
- https://aws.amazon.com/blogs/devops/parallel-and-dynamic-saas-deployments-with-cdk-pipelines/