AWS re:Invent

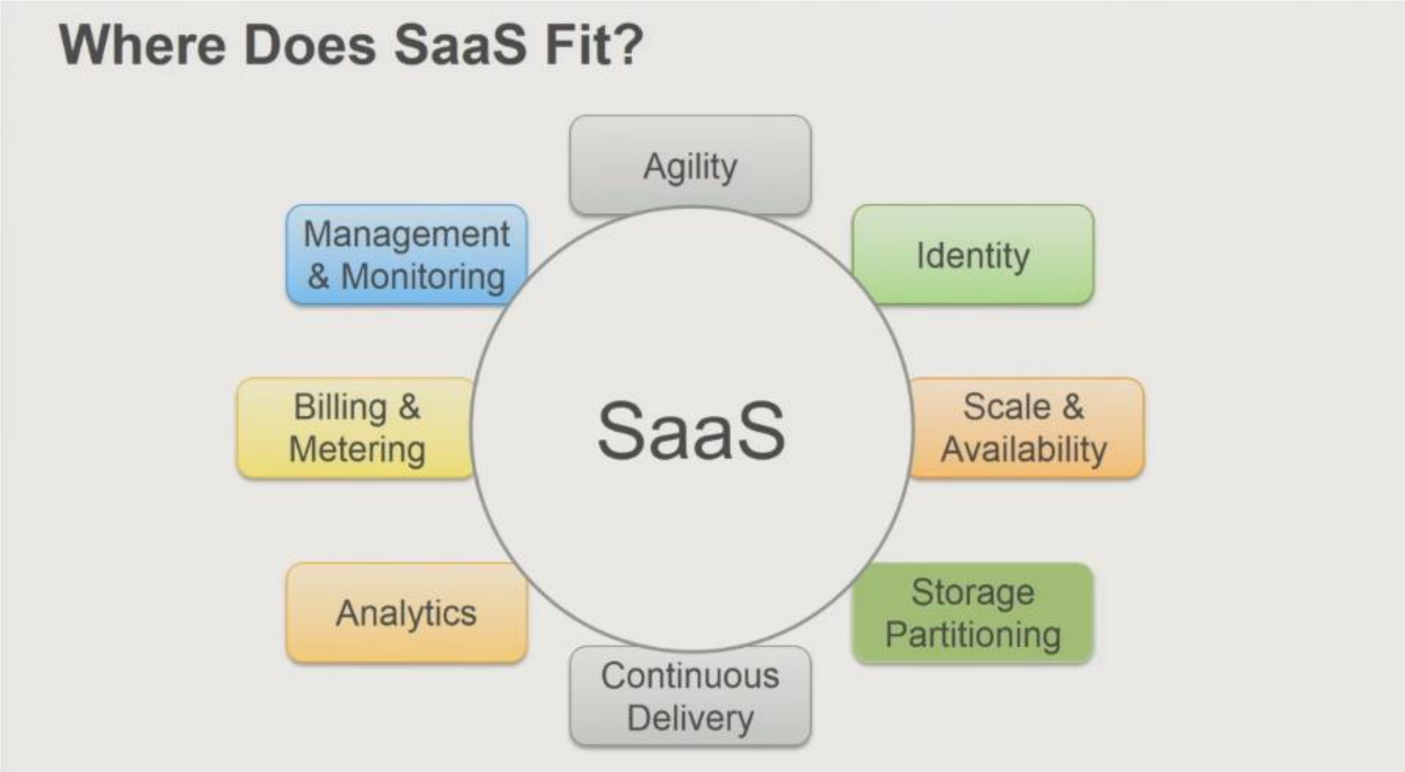
ARC 301

Architecting Next Generation SaaS Applications on AWS

Tod Golding, Partner Solutions Architect

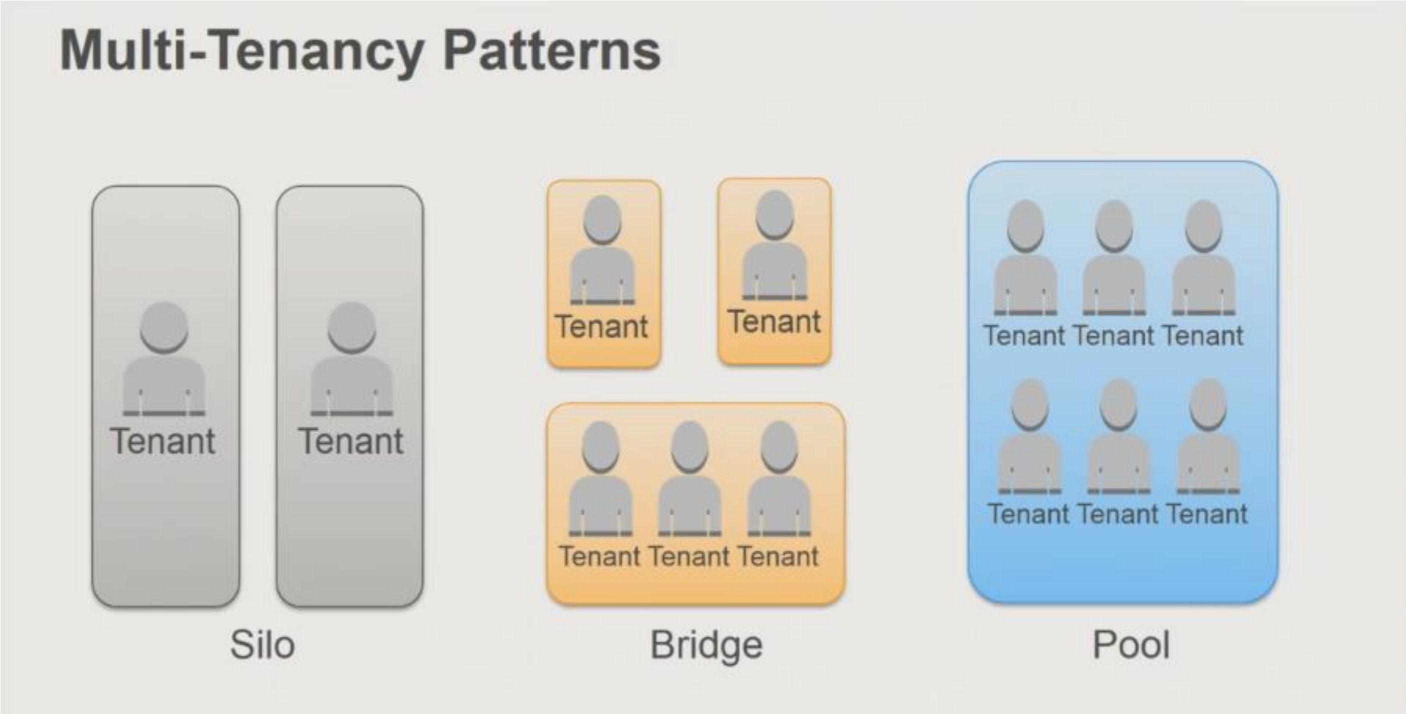
November 2016

AWS provides a broad array of services, tools, and constructs that can be used to design, operate, and deliver SaaS applications. In this session, Tod Golding, the AWS Partner Solutions Architect, shares the wisdom and lessons learned from working with dozens of customers and partners building SaaS solutions on AWS. We discuss key architectural strategies and patterns that are used to deliver multi-tenant SaaS models on AWS and dive into the full spectrum of SaaS design and architecture considerations, including tenant isolation models, tenant identity management, serverless SaaS, and multi-tenant storage strategies. This session connects the dots between general SaaS best practices and what it means to realize these patterns on AWS, weighing the architectural tradeoffs of each model and assessing its influence on the agility, manageability, and cost profile of your SaaS solution.



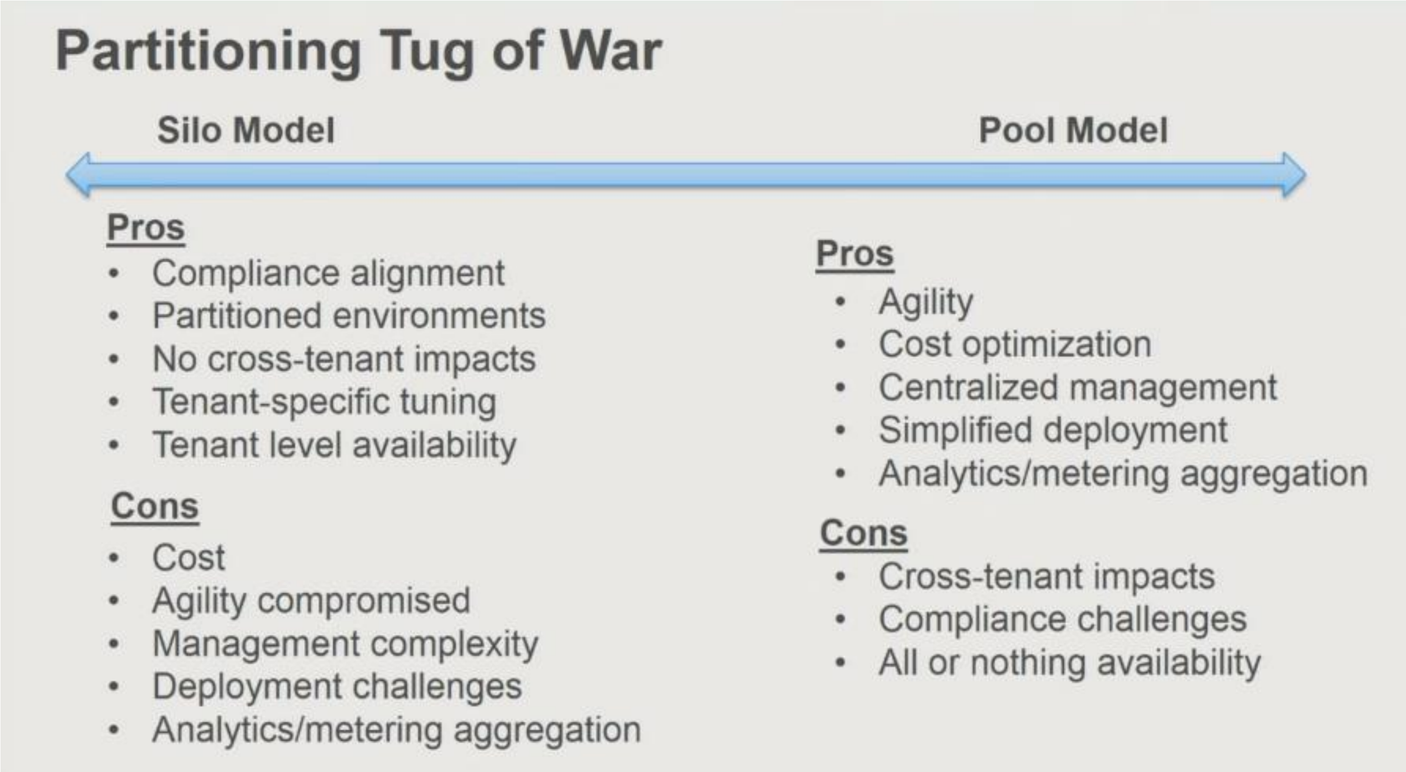
Where Does SaaS Fit?

What is the best SaaS delivery model on AWS? SaaS intersects each one of the architectural concepts shown above.



Multi-Tenancy Patterns

SaaS is all about how fast can we add features for all our multi-tenants in our platform. What are the architectural patterns that we can leverage? How does AWS and the services affect how I approach these patterns? ***Silo*** is all about putting each one of our tenants in a fully isolated infrastructure, with their own underlying infra and footprint and be entirely isolated. We then bolt on top of these onboarding and some sharing functionalities to create a seamless experience, but the will have fully separate infrastructure. The ***Bridge*** model is a hybrid model where we are going to have some multi-tenant approach for some cases and in other we will have the silo for each users. The ***Pool*** is where we share all facilities between all our users.



Partitioning Tug of War

Silo Model

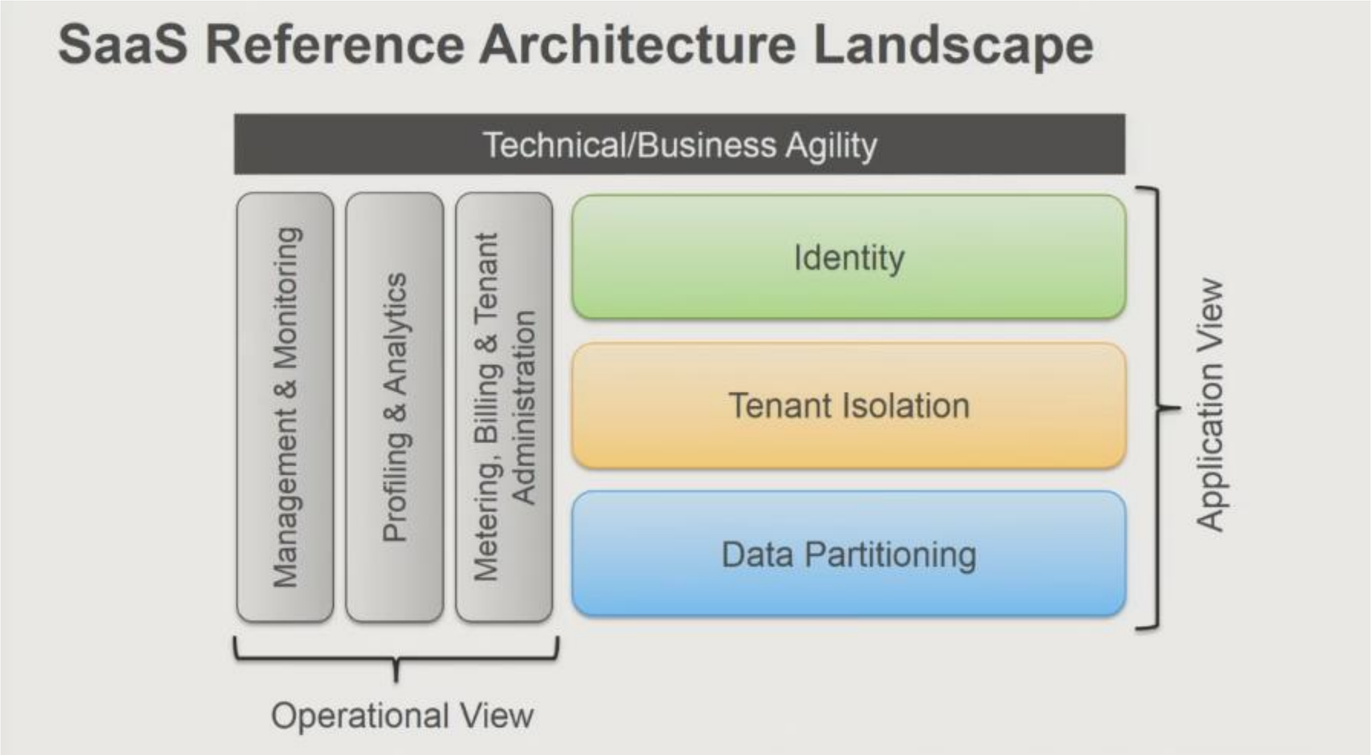
Pros
- Compliance alignment
- Partitioned environments
- No cross-tenant impacts
- Tenant-specific tuning
- Tenant level availability

Cons
- Cost
- Agility compromised
- Management complexity
- Deployment challenges
- Analytics/metering aggregation

Pool Model

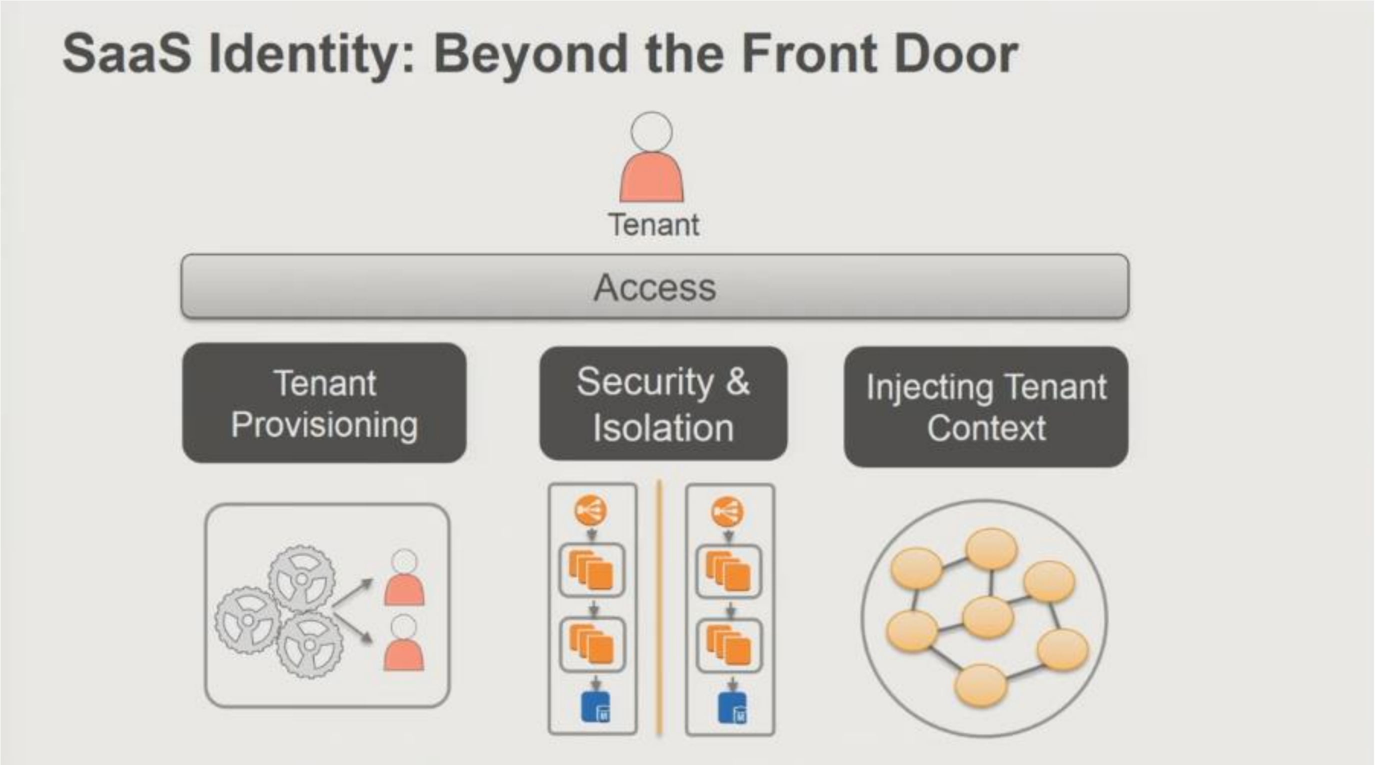
Pros
- Agility
- Cost optimization
- Centralized management
- Simplified deployment
- Analytics/metering aggregation

Cons
- Cross-tenant impacts
- Compliance challenges
- All or nothing availability



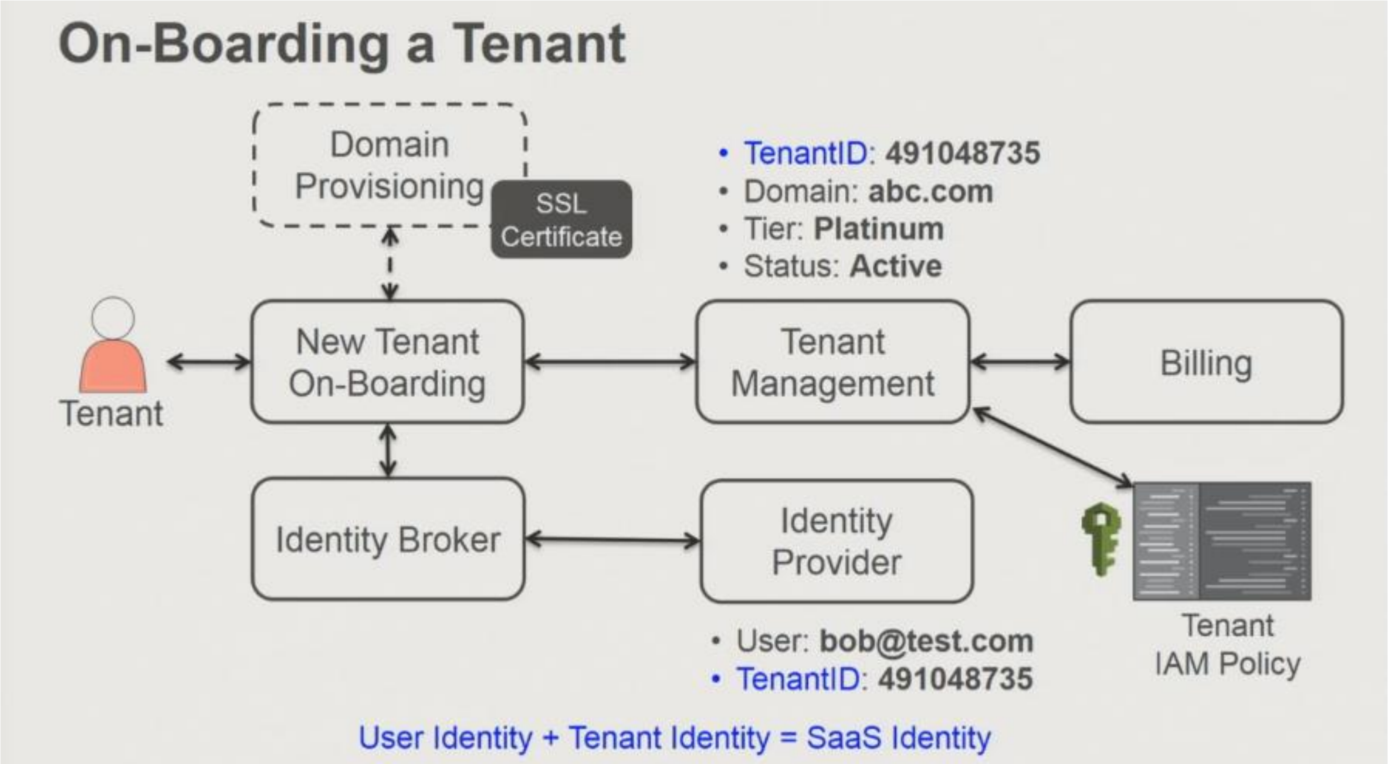
SaaS Reference Architecture Landscape

The Application View are the things you need to do to structure your application, AWS offers you a wide array of ways to separate tenants, data partitioning deals with how we separate tenant data and what does multi-tenancy look like using some of the database offering. Operational View deals with how we help the customer do things like monitoring and management. The Technical/Business Agility influences everything. In SaaS organizations, the agility is important when trying to add new features quickly and how quickly we can change our architecture to meet new or changing demands.



SaaS Identity: Beyond the Front Door

Identity goes beyond AuthO and AuthZ, it includes how our tenants can have security and access their data and resources securely using identity or making the resources only available to specific tenants in a multi-tenant environment. The notion of tenant context has to flow through all the microservices in the SaaS platform.



On-Boarding a Tenant

Domain Provisioning

SSL Certificate

- TenantID: 491048735
- Domain: abc.com
- Tier: Platinum
- Status: Active

Tenant

New Tenant On-Boarding

Tenant Management

Billing

Identity Broker

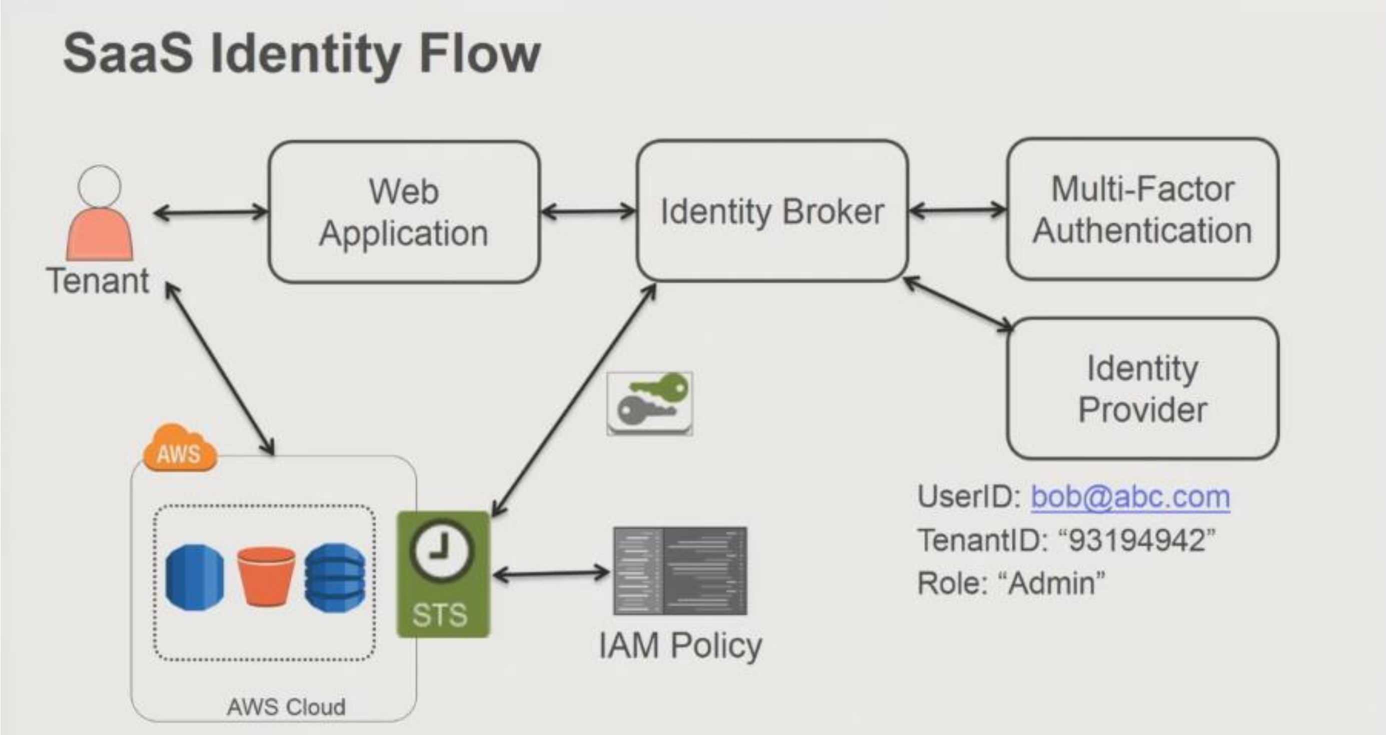
Identity Provider

Tenant IAM Policy

- User: bob@test.com
- TenantID: 491048735

User Identity + Tenant Identity = SaaS Identity

You need to get the user profile created during basic sign up and validation and then get the tenant profile created to actually create the tenant landscape and get a 3rd party service setup for things like billing. The last part is the creation of the IAM policy and Roles that will help scope the tenant's access to the actual resources.



SaaS Identity Flow

Tenant

Web Application

Identity Broker

Multi-Factor Authentication

Identity Provider

AWS

AWS Cloud

STS

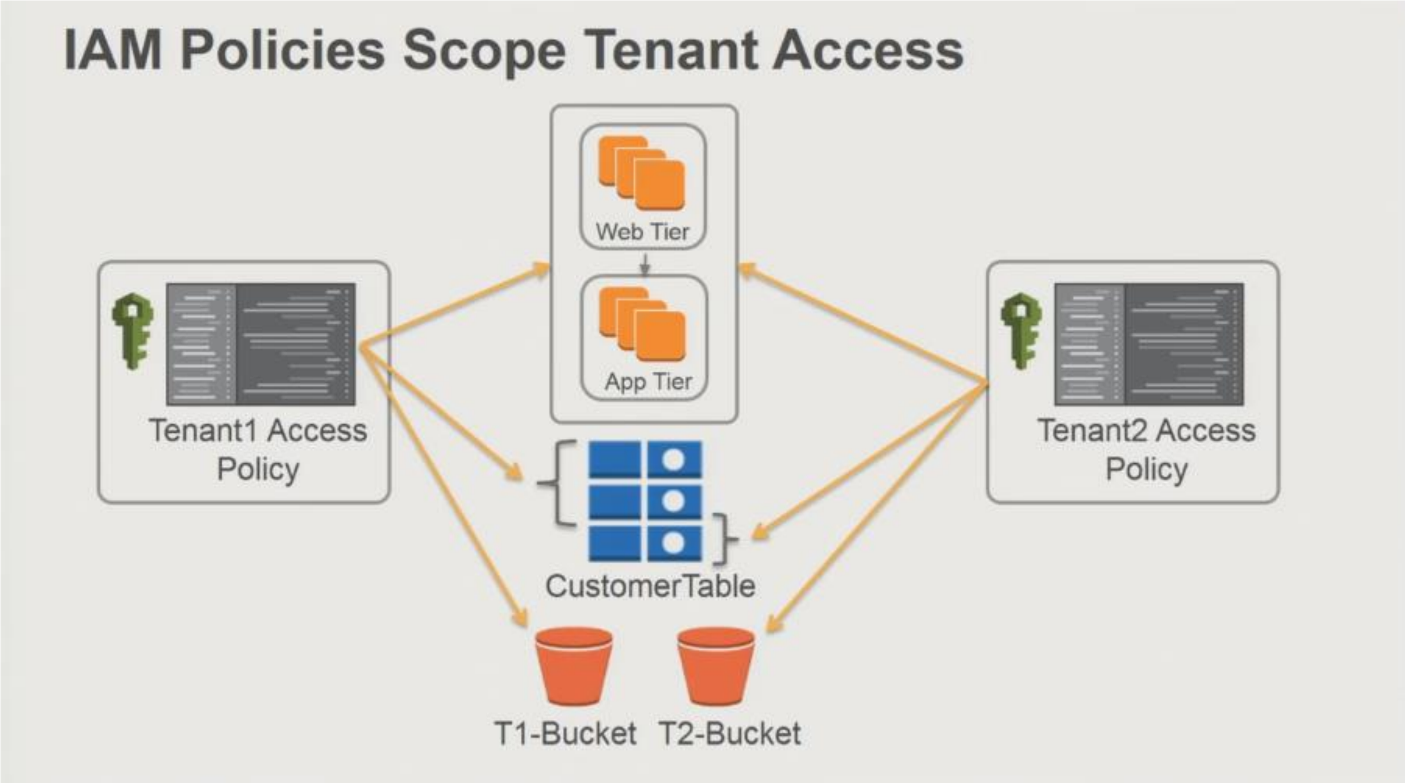
IAM Policy

UserID: bob@abc.com

TenantID: "93194942"

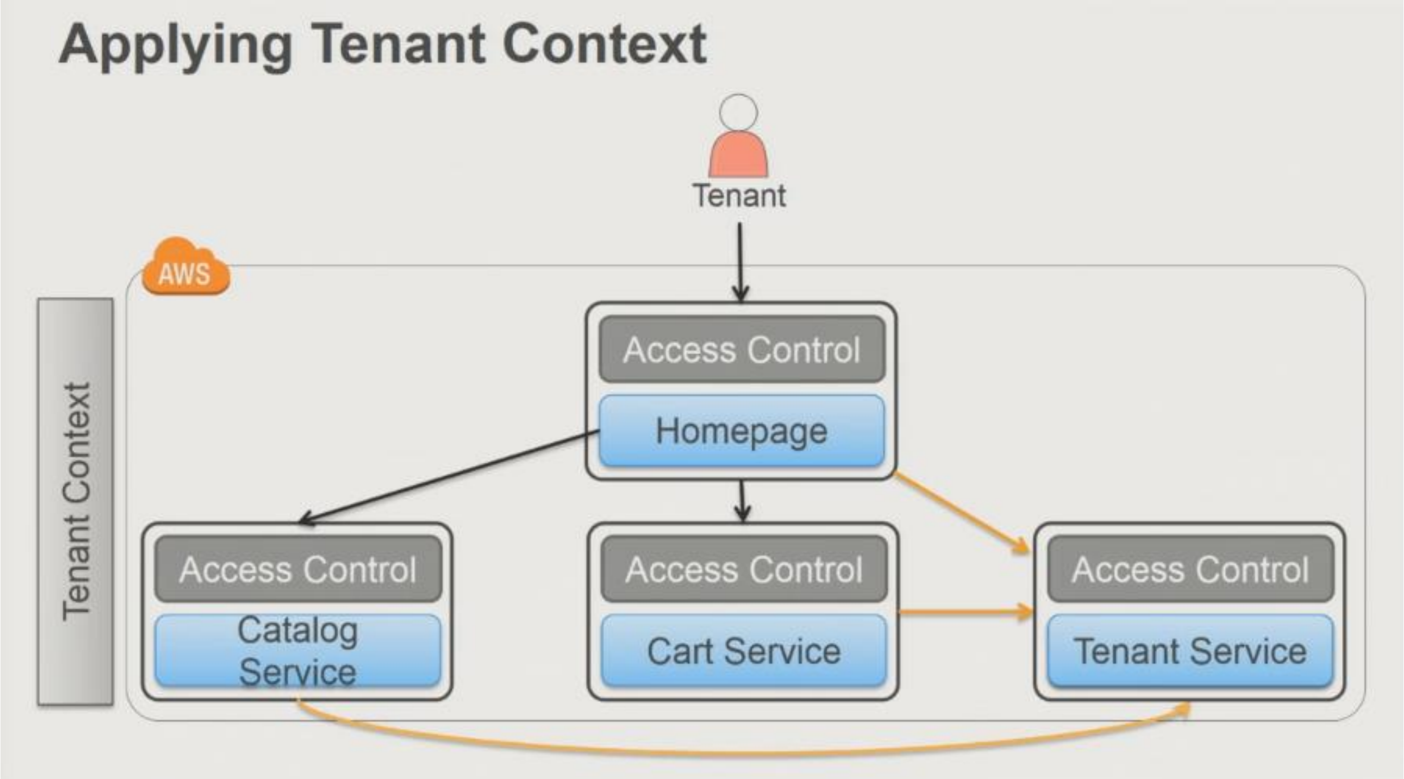
Role: "Admin"

When the user gets on the page and gets redirected to the identity provider to login and get an identity token, the user has not really solved the tenant SaaS identity part of the problem yet because this has not constrained the user's view of the SaaS application's landscape. We can now use the Roles and IAM Policy that we provisioned in the prior step, we can then use the STS service from AWS to get an assumed role with a web identity, we take the id token and the role (with a set of policies we created for this tenant) the tenant wants to assume to do some functions and go get the needed temporary credentials that will control and manage what the user can do or not do with AWS services and resources.



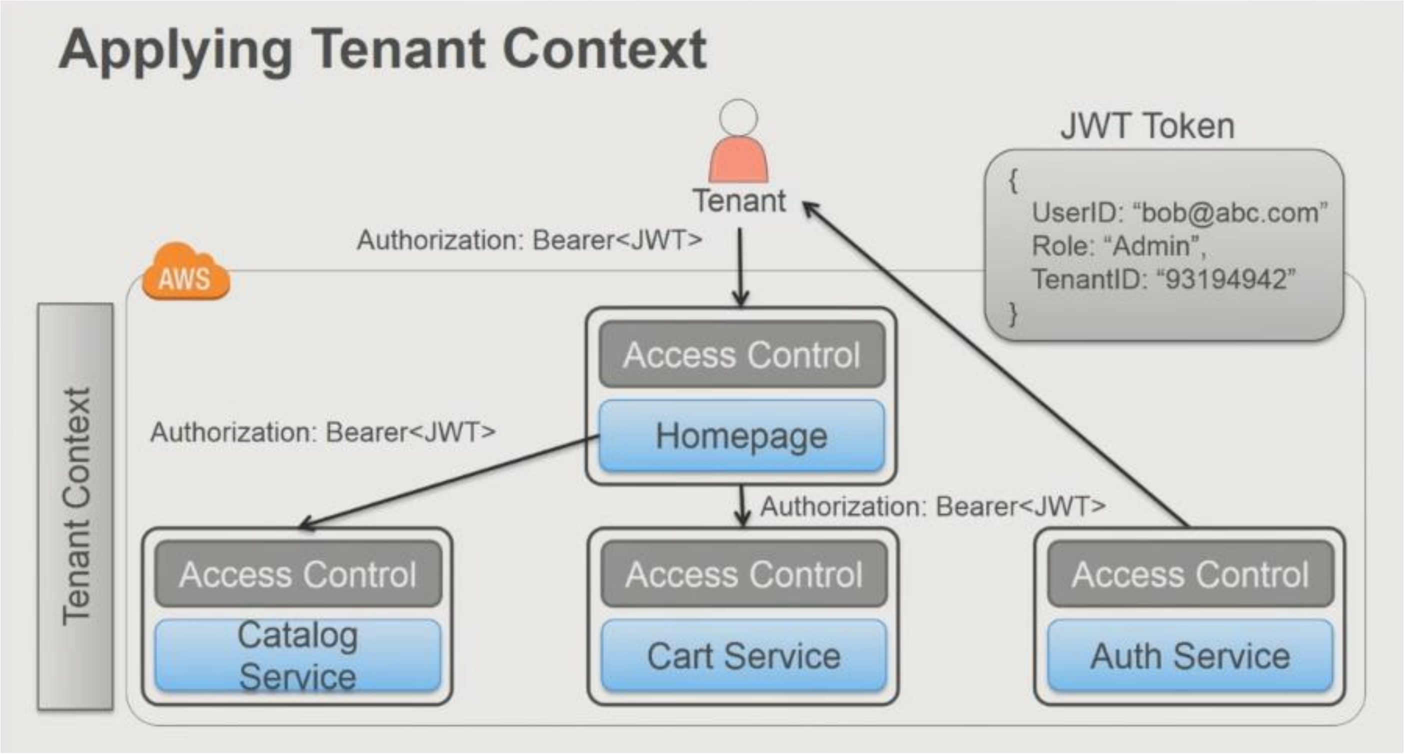
IAM Policies Scope Tenant Access

We can apply IAM role at different granularity to scope and control access to resources like tables, buckets, etc.



Applying Tenant Context

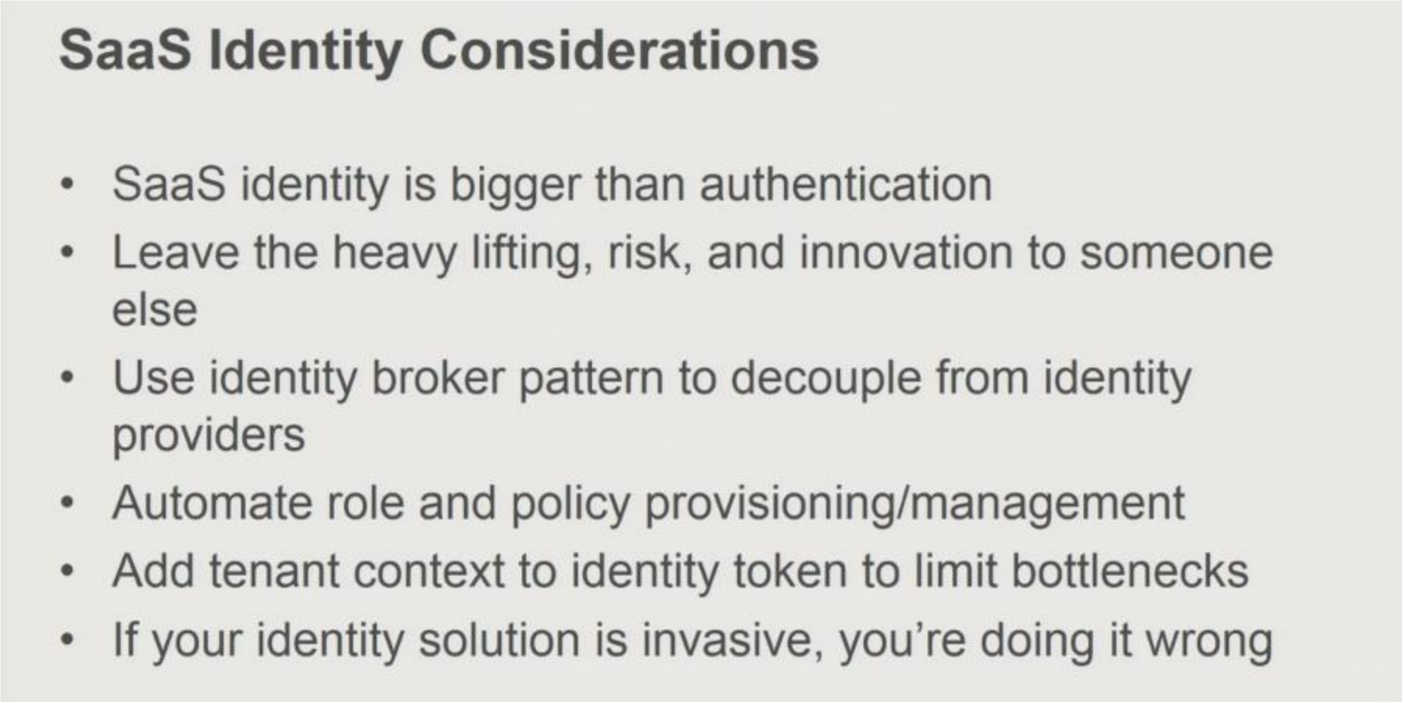
The Tenant service being used in this approach to resolve the context quickly becomes a bottleneck when we start having many microservices in the platform.



Applying Tenant Context

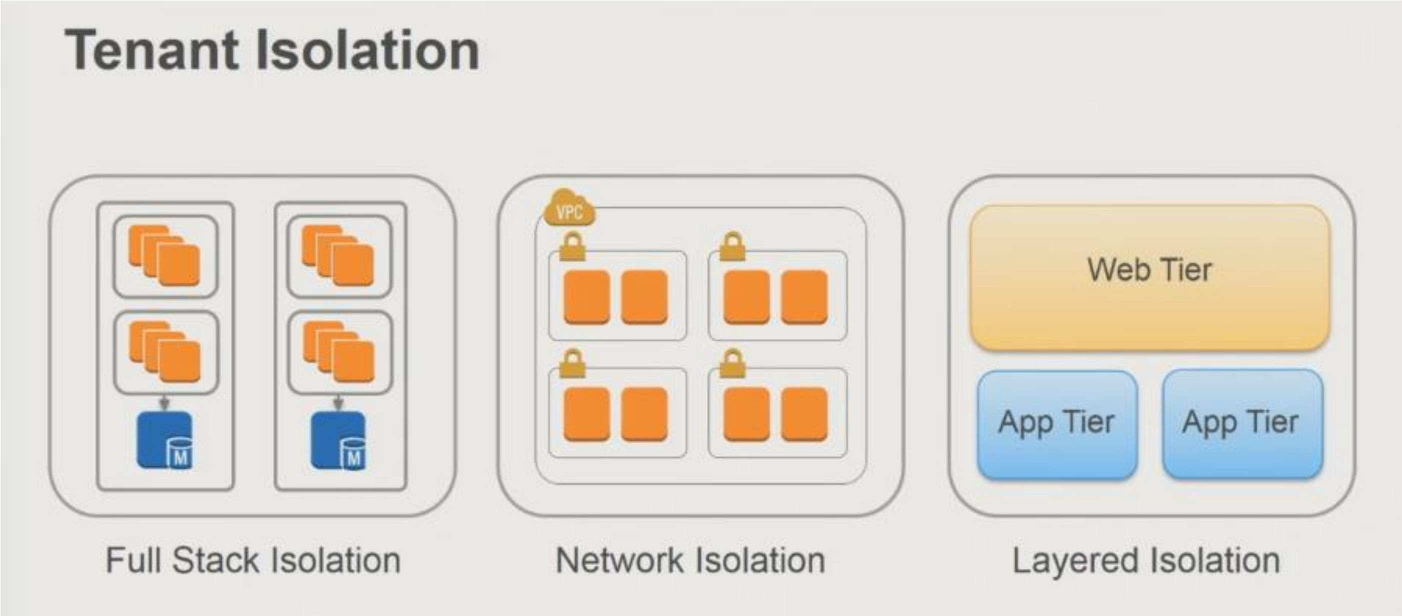
```
{
  UserID: "bob@abc.com"
  Role: "Admin",
  TenantID: "93194942"
}
```

We can instead introduce an Auth service (like Auth0, Okta, etc) that we can authenticate with and get a JWT token that has the necessary claims in it, we configure that JWT token with the tenant's true SaaS identity UserID, TenantID, and Role. This JWT flows with the tenant across all the services they request to use.



SaaS Identity Considerations

- SaaS identity is bigger than authentication
- Leave the heavy lifting, risk, and innovation to someone else
- Use identity broker pattern to decouple from identity providers
- Automate role and policy provisioning/management
- Add tenant context to identity token to limit bottlenecks
- If your identity solution is invasive, you're doing it wrong



Tenant Isolation

VPC

Web Tier

App Tier

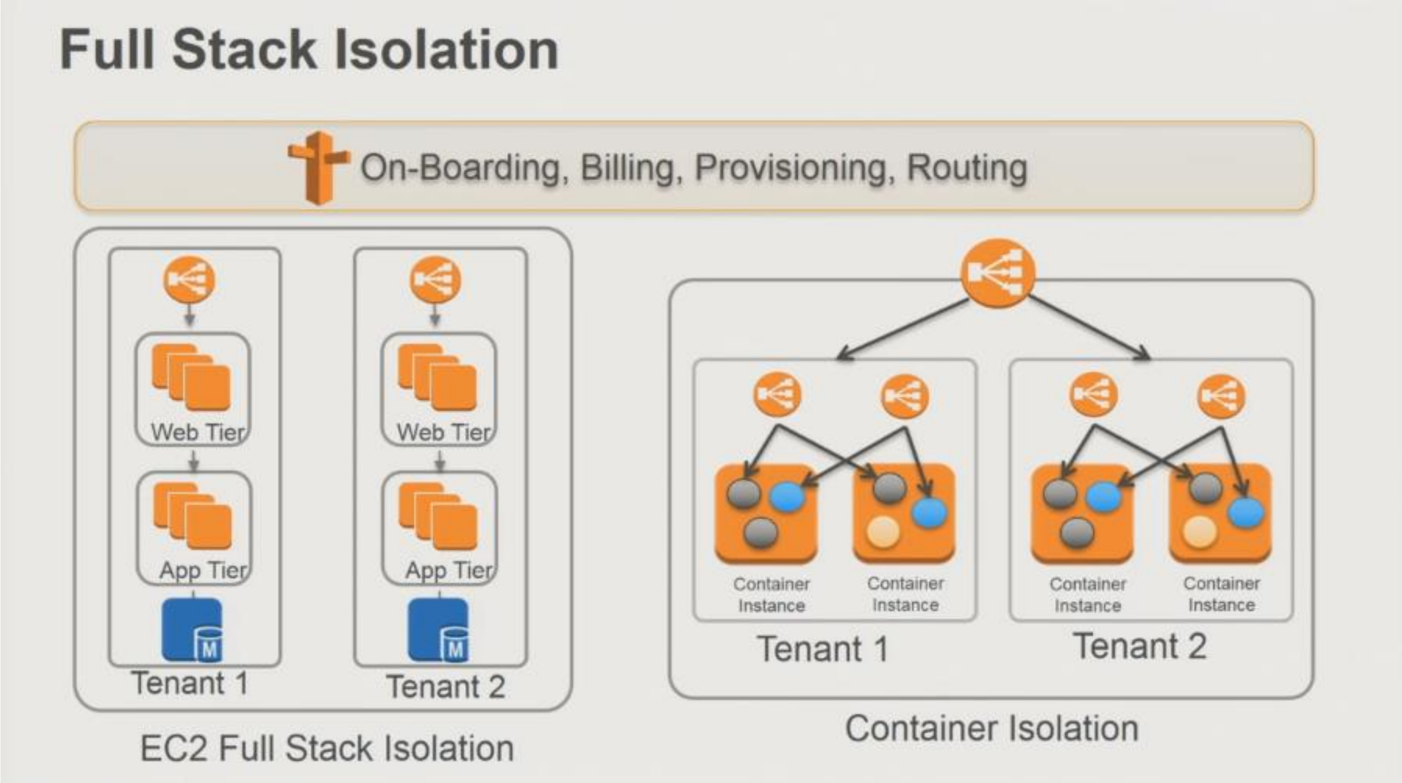
App Tier

Full Stack Isolation

Network Isolation

Layered Isolation

These are the isolation approaches we can use to isolate tenants in AWS.



Full Stack Isolation

On-Boarding, Billing, Provisioning, Routing

Web Tier

App Tier

Tenant 1

Web Tier

App Tier

Tenant 2

EC2 Full Stack Isolation

Container Instance

Container Instance

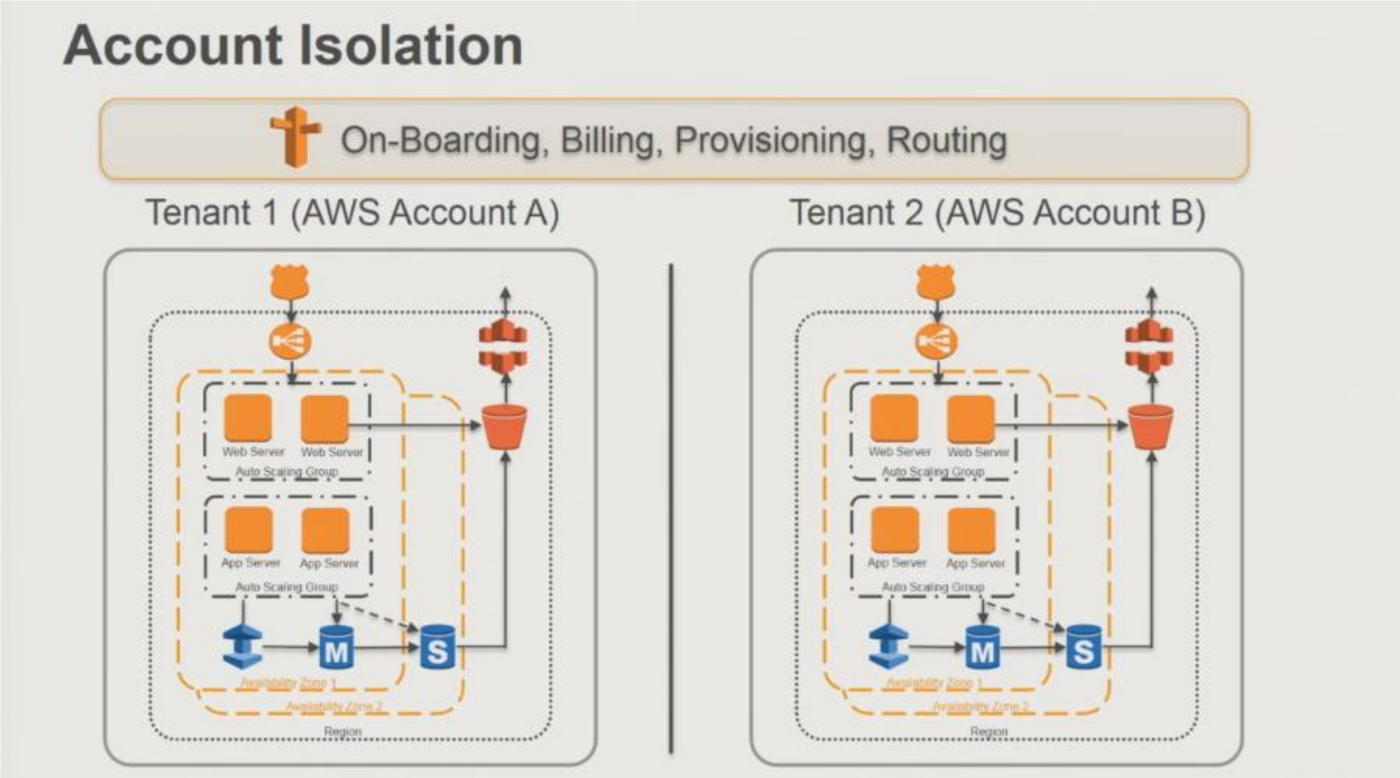
Tenant 1

Container Instance

Container Instance

Tenant 2

Container Isolation



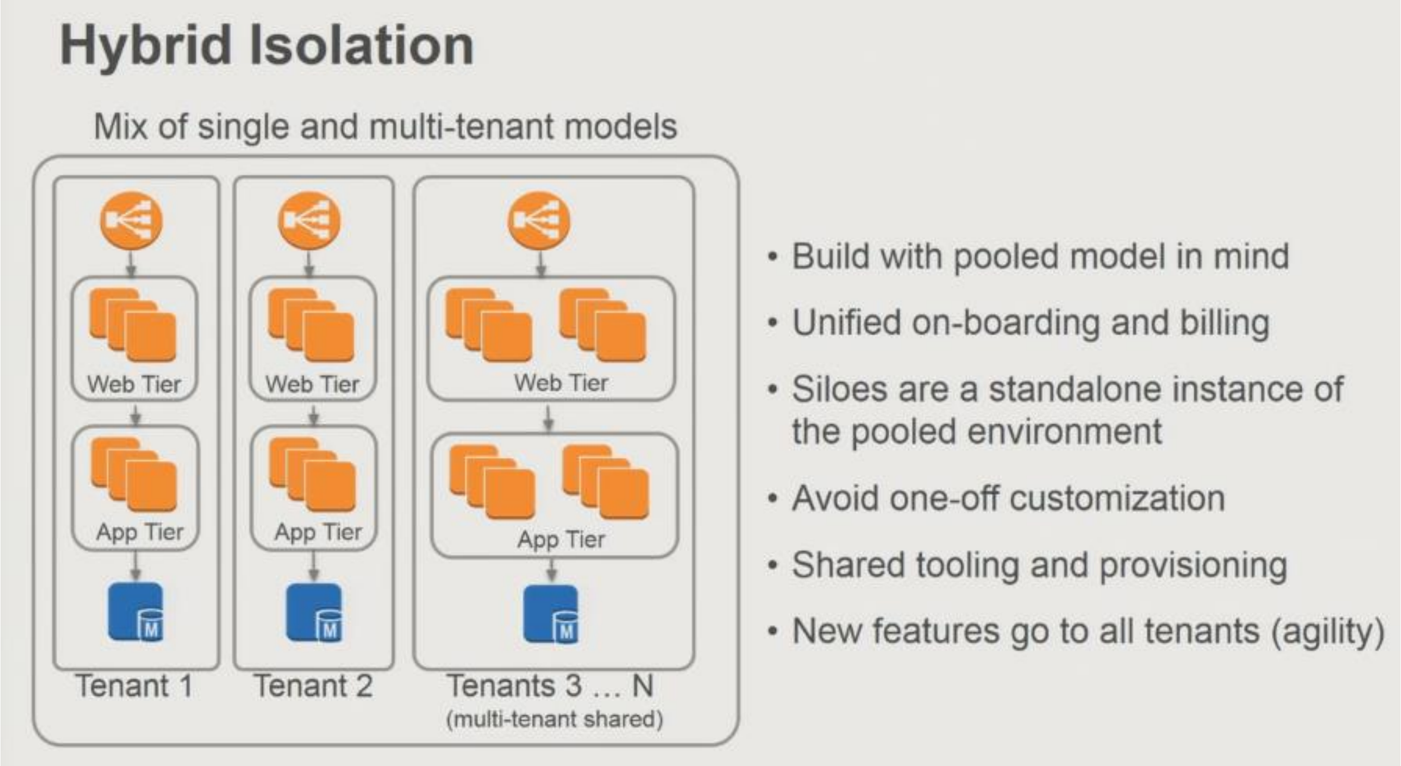
Account Isolation

On-Boarding, Billing, Provisioning, Routing

Tenant 1 (AWS Account A)

Tenant 2 (AWS Account B)

This approach uses AWS linked accounts where we have a payer account and several child accounts for each tenant. We can see the child consumption as a separate tab in our AWS monthly bill too. This is not scalable for 1,000s of tenants that need individual child accounts

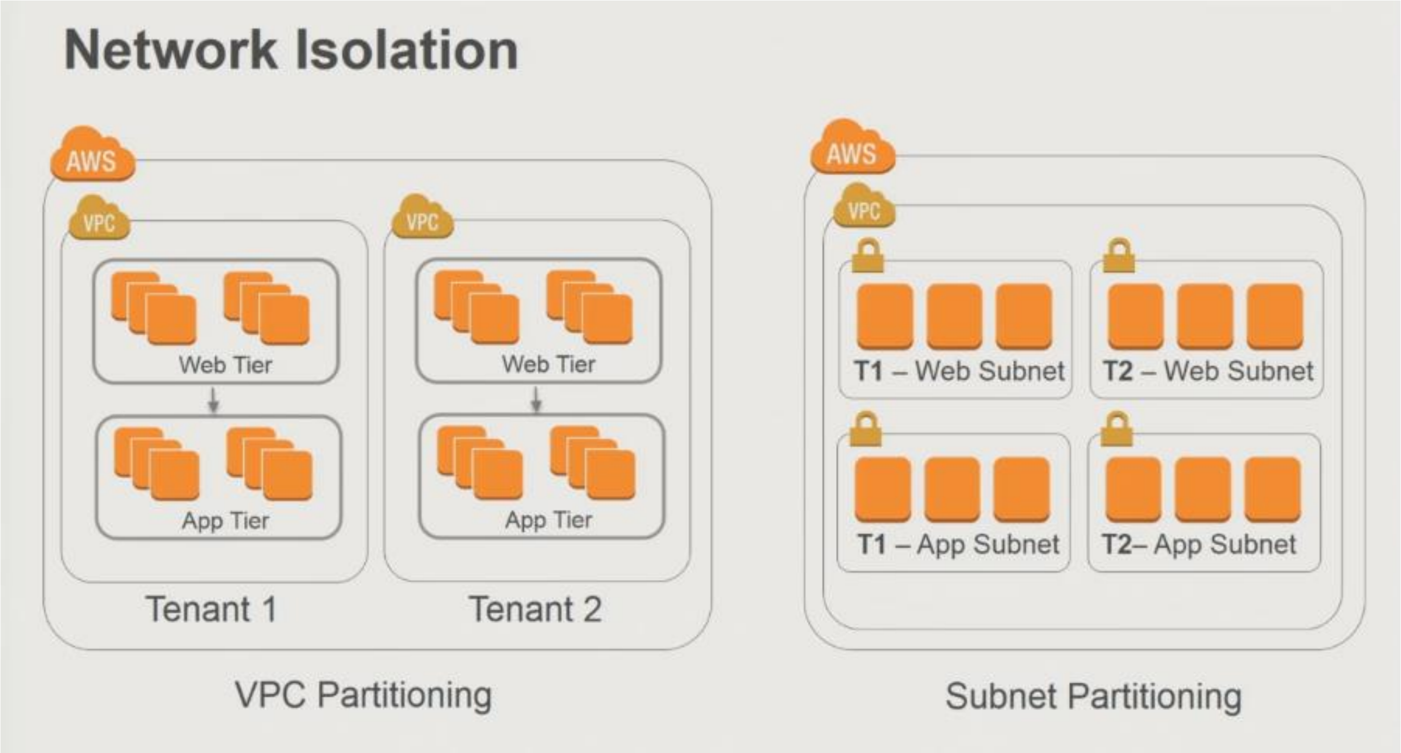


Hybrid Isolation

Mix of single and multi-tenant models

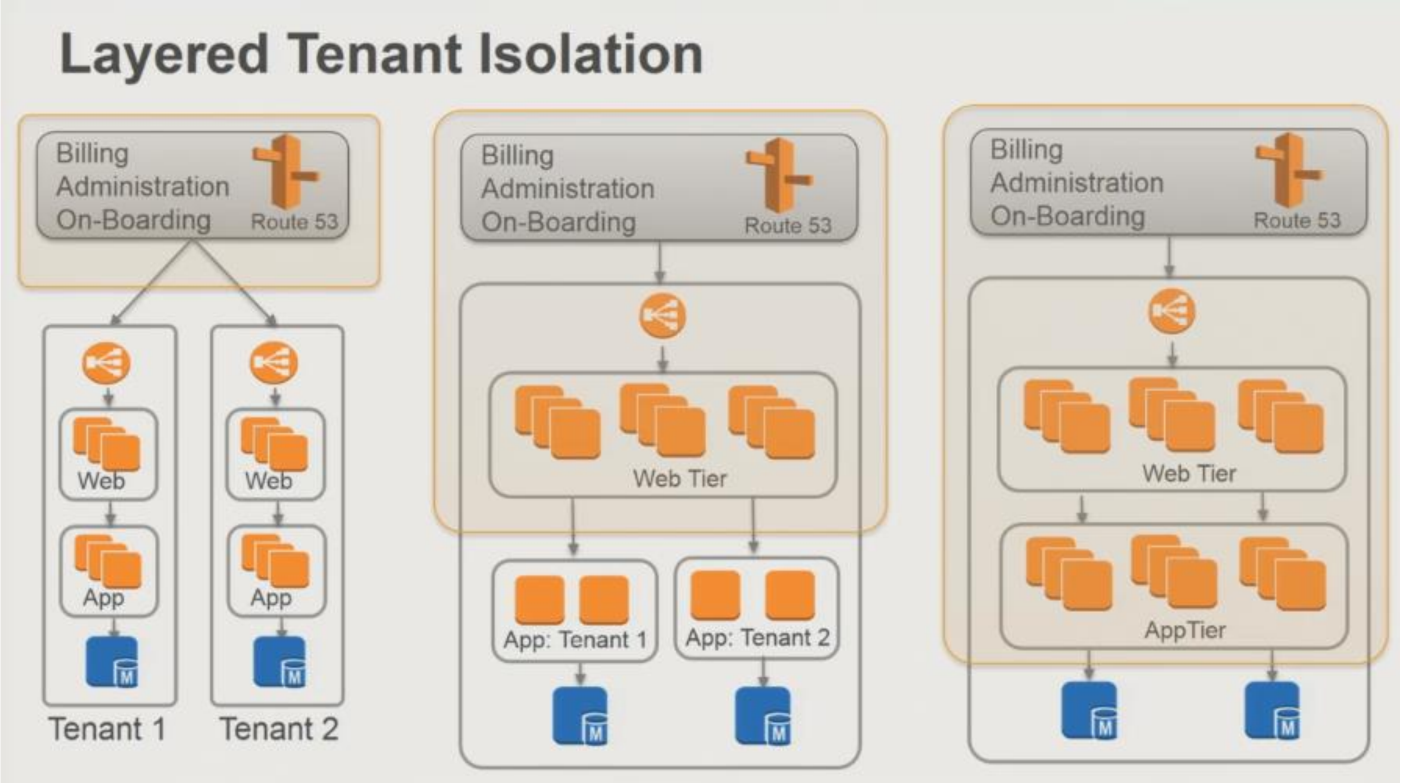
- Build with pooled model in mind
- Unified on-boarding and billing
- Siloes are a standalone instance of the pooled environment
- Avoid one-off customization
- Shared tooling and provisioning
- New features go to all tenants (agility)

Start with a shared environment and then create isolated environments for tenants that specifically require that, we would still keep the same DevOps tooling and setup that we used for the shared environment.



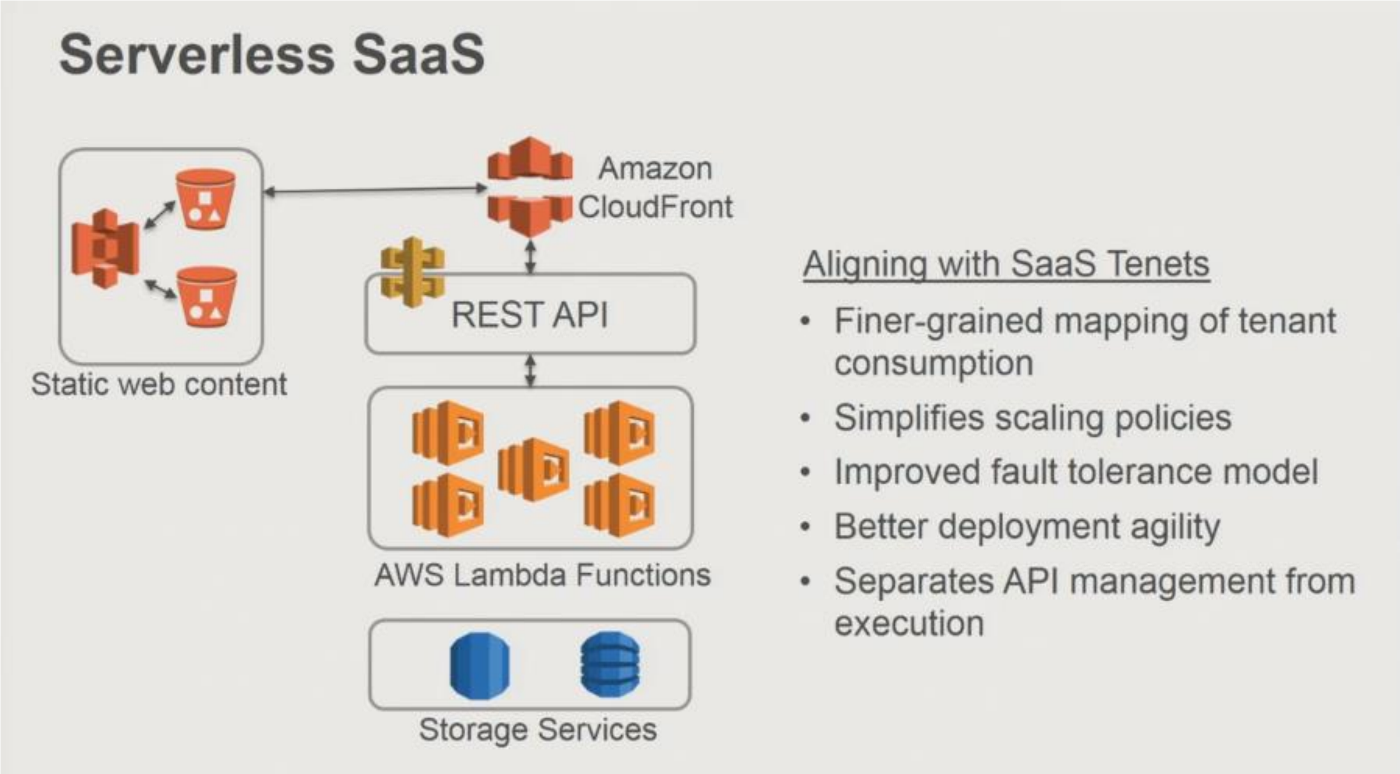
Network Isolation

We can create a VPC per tenant and then peer those VPCs to get the wider picture, we now have to use tagging to link resources with tenants. We can also create separate subnets for each tenants too and use tagging extensively.



Layered Tenant Isolation

We can slowly move from the left to the right over a period of time from isolated to fully multi-tenancy.

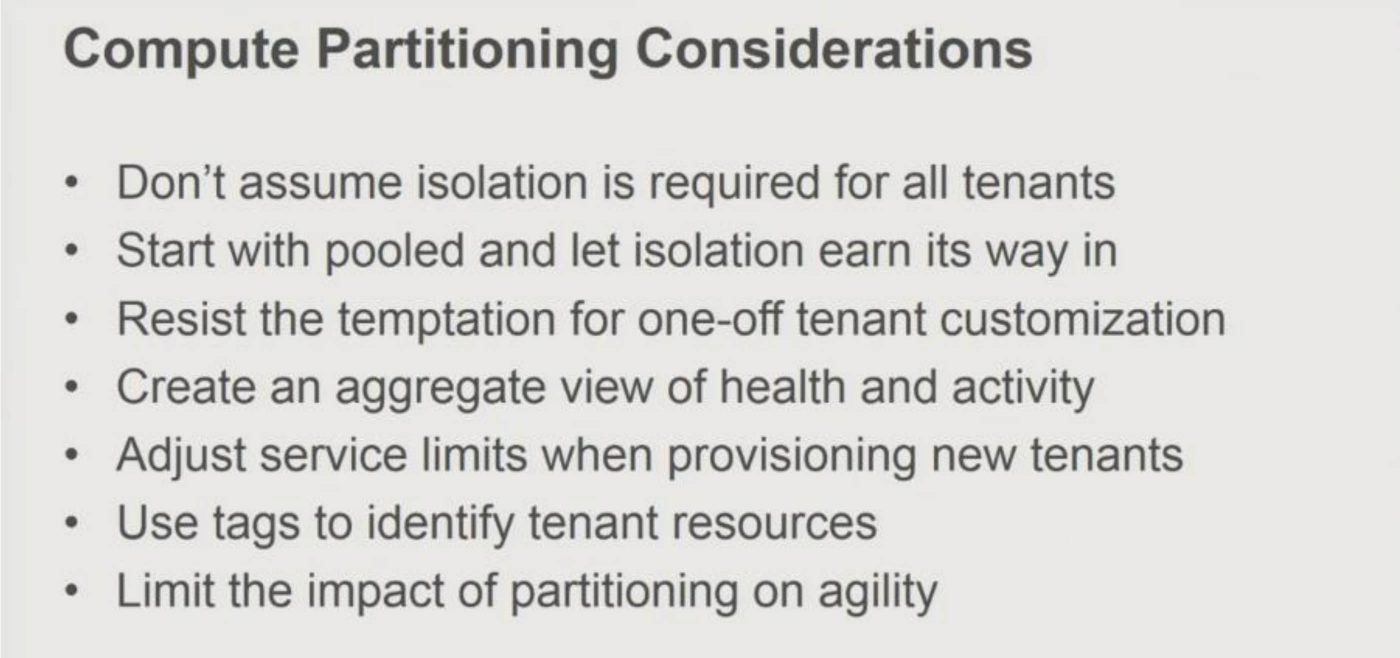


Serverless SaaS

Aligning with SaaS Tenets

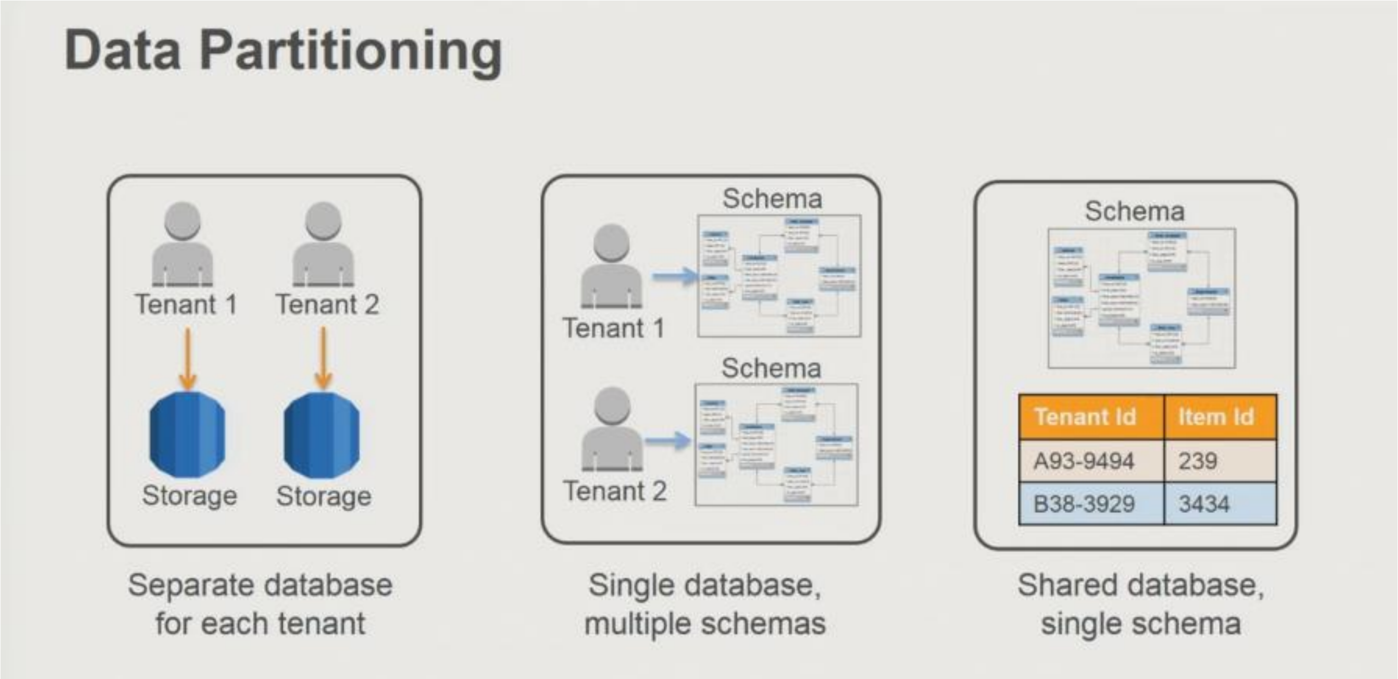
- Finer-grained mapping of tenant consumption
- Simplifies scaling policies
- Improved fault tolerance model
- Better deployment agility
- Separates API management from execution

API Gateway can provide throttling, metering, filtering, etc and we can have lambdas for each thing our application does. We don’t need entire stacks for each tenant and instead use different contexts for each tenant when running certain functions. This is the best match for infrastructure and tenant consumption.



Compute Partitioning Considerations

- Don’t assume isolation is required for all tenants
- Start with pooled and let isolation earn its way in
- Resist the temptation for one-off tenant customization
- Create an aggregate view of health and activity
- Adjust service limits when provisioning new tenants
- Use tags to identify tenant resources
- Limit the impact of partitioning on agility



Data Partitioning

Tenant 1 Tenant 2

Storage Storage

Separate database for each tenant

Schema

Tenant 1

Schema

Tenant 2

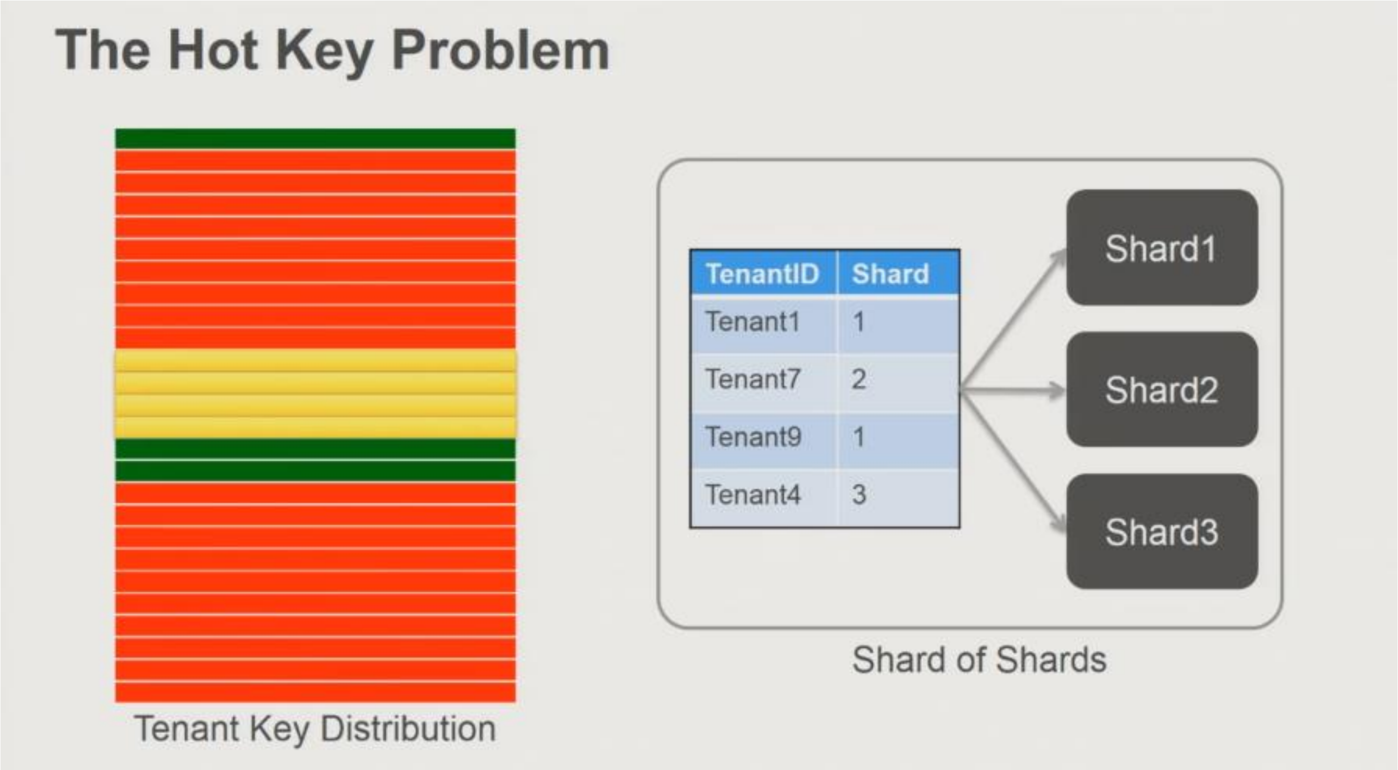
Single database, multiple schemas

Schema

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

Shared database, single schema

These are the 3 flavors of data partitioning.



The Hot Key Problem

Tenant Key Distribution

TenantID	Shard
Tenant1	1
Tenant7	2
Tenant9	1
Tenant4	3

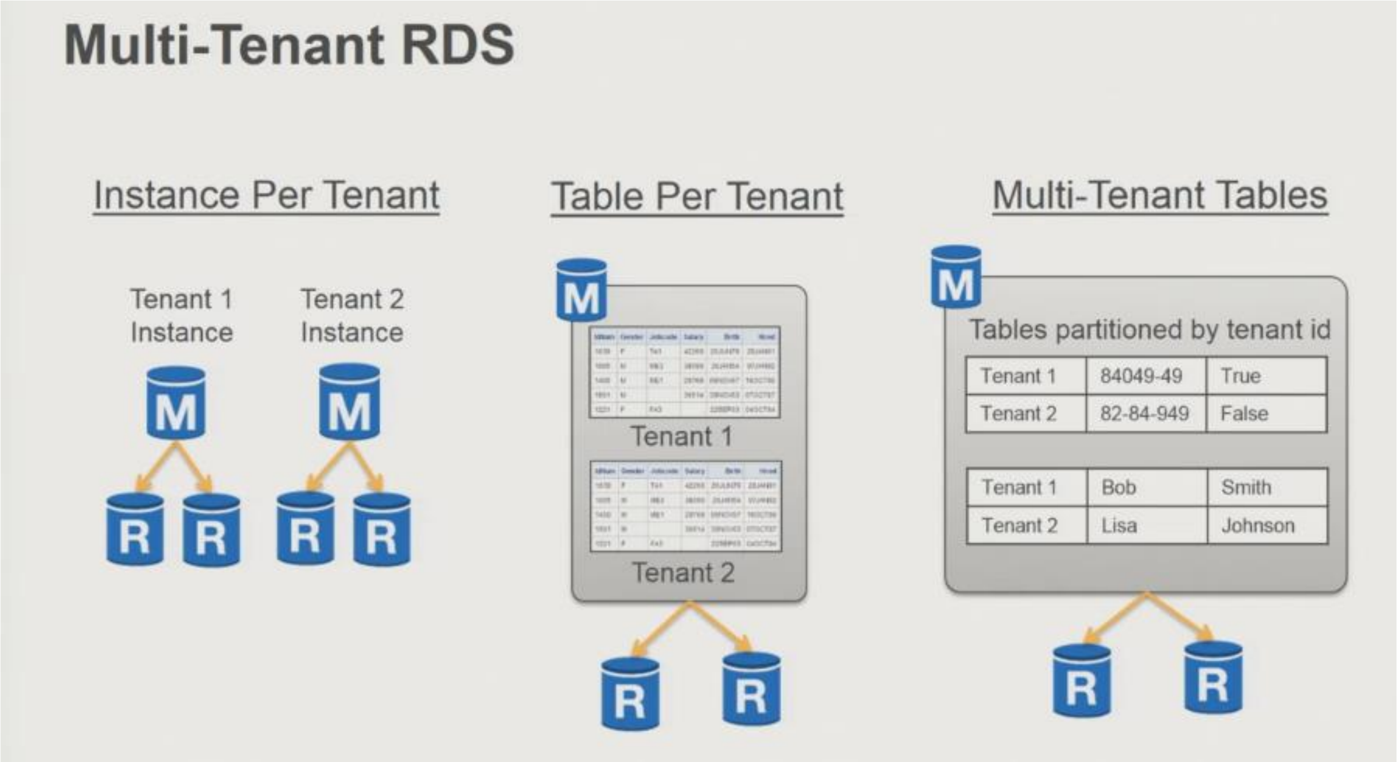
Shard1

Shard2

Shard3

Shard of Shards

The distribution of data in a shared environment is rarely even and the big data tenants can affect the distribution of the keys on your sharding scheme. You need to figure out how the data can be sharded effectively for each tenant to overcome the data distribution problem.



Multi-Tenant RDS

Instance Per Tenant

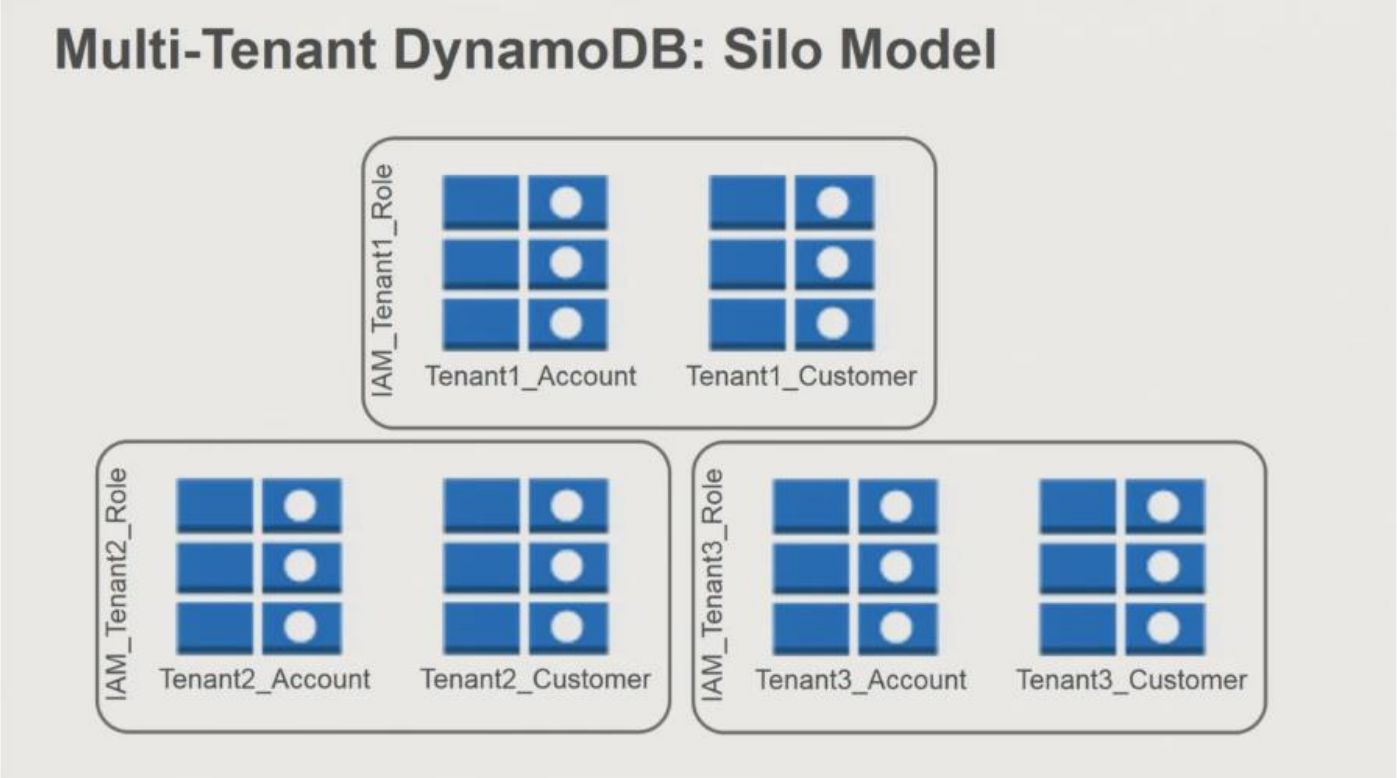
Table Per Tenant

Multi-Tenant Tables

Tables partitioned by tenant id

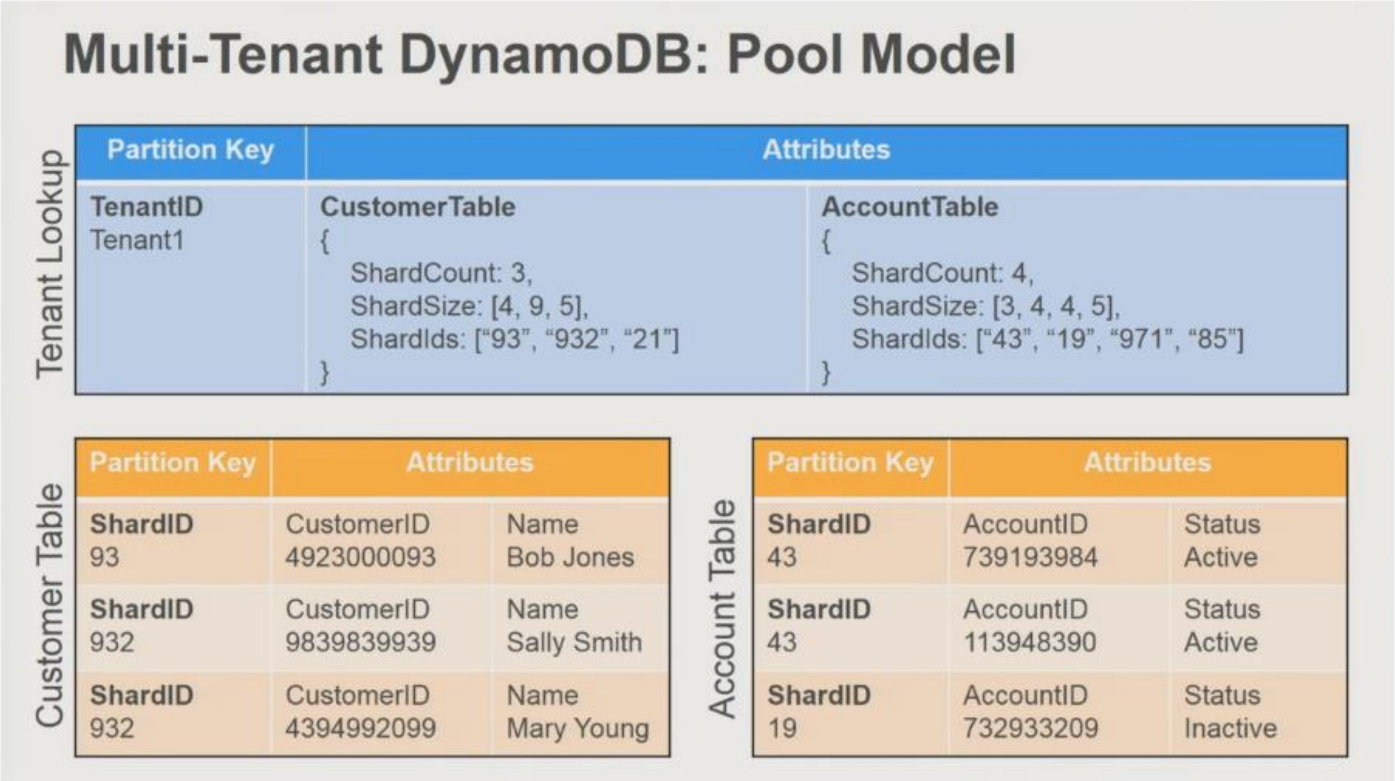
Tenant 1	84049-49	True
Tenant 2	82-84-949	False

Tenant 1	Bob	Smith
Tenant 2	Lisa	Johnson



Multi-Tenant DynamoDB: Silo Model

We can create tables for each tenant and scope them with the IAM roles for that tenant to be able to access the table



Multi-Tenant DynamoDB: Pool Model

Tenant Lookup

Partition Key	Attributes	

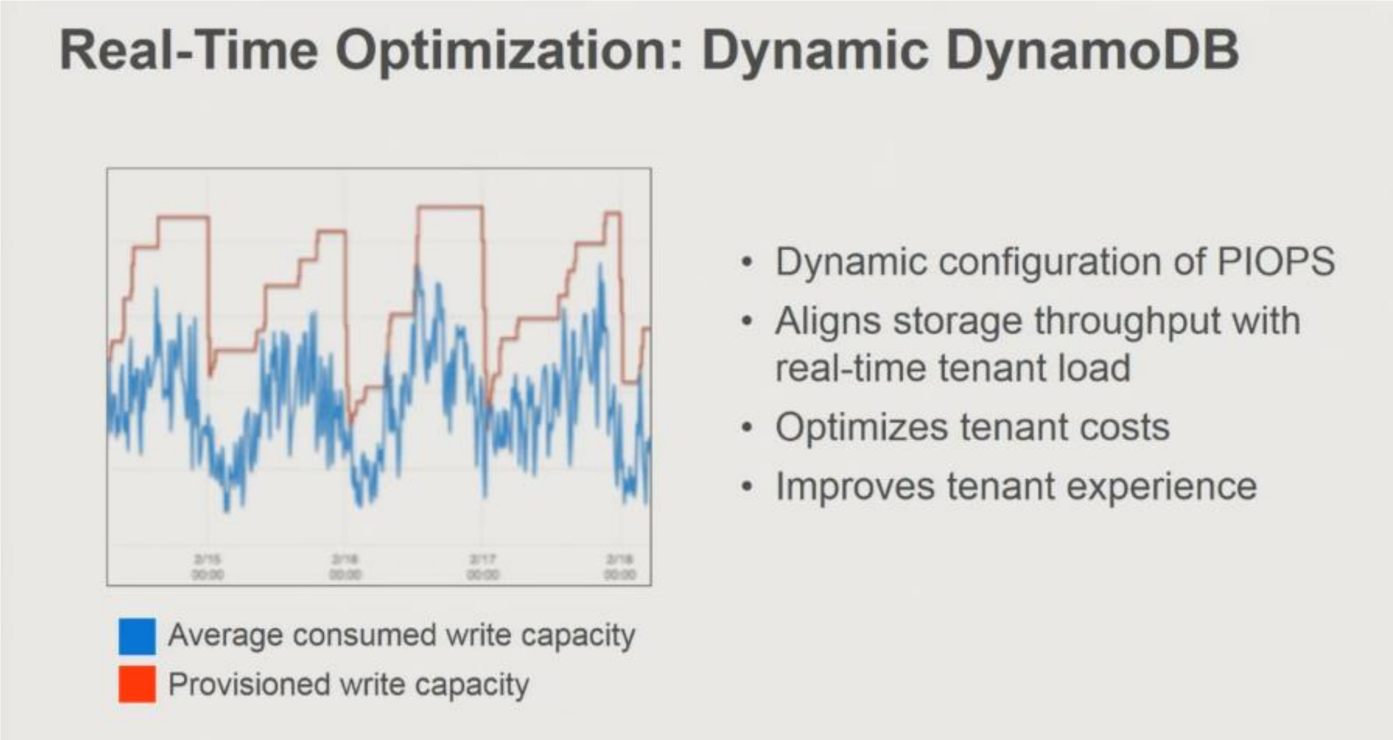
TenantID
Tenant1
	CustomerTable
	{
	ShardCount: 3,
	ShardSize: [4, 9, 5],
	ShardIds: [“93”, “932”, “21”]
	}
		AccountTable
		{
		ShardCount: 4,
		ShardSize: [3, 4, 4, 5],
		ShardIds: [“43”, “19”, “971”, “85”]
		}

Customer Table

Partition Key	Attributes	
ShardID 93	CustomerID 4923000093	Name Bob Jones
ShardID 932	CustomerID 9839839939	Name Sally Smith
ShardID 932	CustomerID 4394992099	Name Mary Young

Account Table

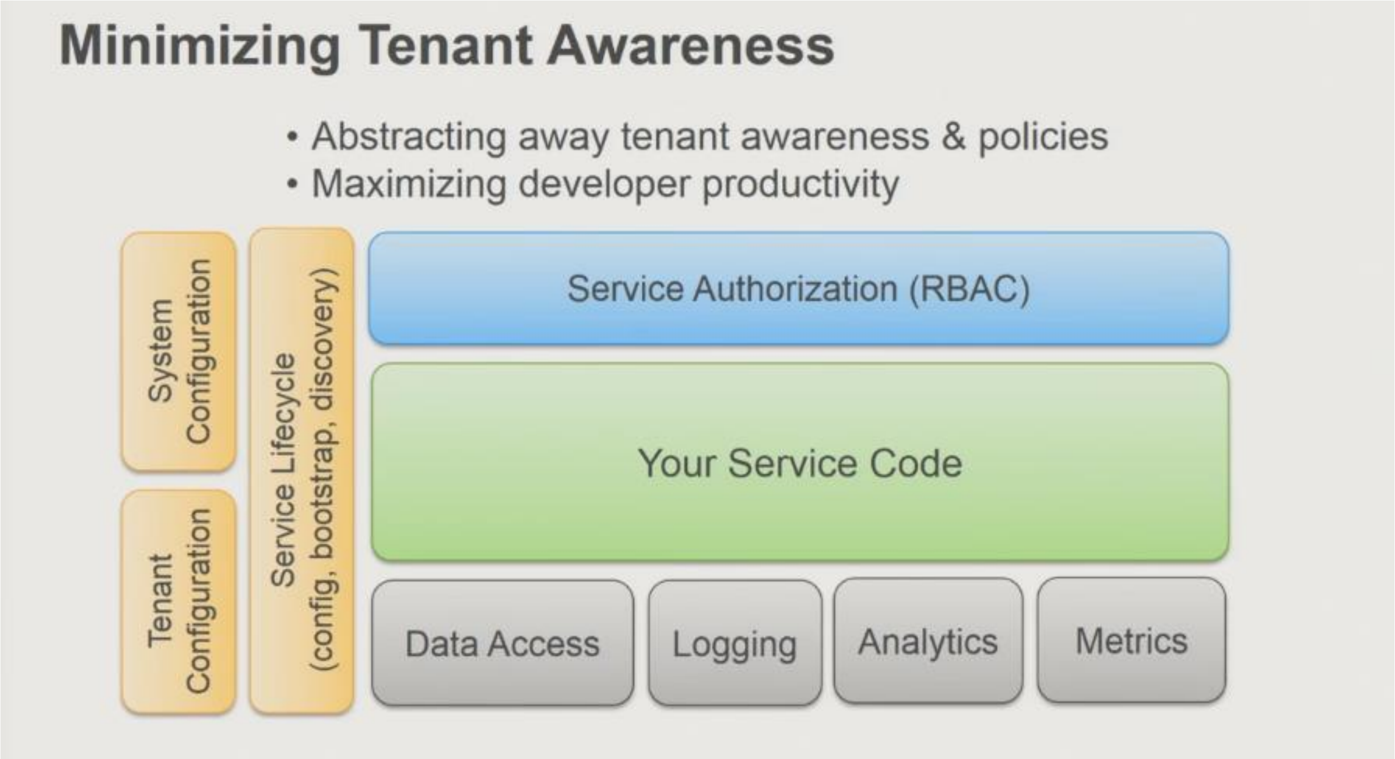
Partition Key	Attributes	
ShardID 43	AccountID 739193984	Status Active
ShardID 43	AccountID 113948390	Status Active
ShardID 19	AccountID 732933209	Status Inactive



Real-Time Optimization: Dynamic DynamoDB

- Dynamic configuration of PIOPS
- Aligns storage throughput with real-time tenant load
- Optimizes tenant costs
- Improves tenant experience

Think about optimizing for real-time using tools like DynamicDynamoDB to help automatically adjust the IOPS for the DynamoDB table as the load changes in real-time



Minimizing Tenant Awareness

- Abstracting away tenant awareness & policies
- Maximizing developer productivity

System Configuration

Tenant Configuration

Service Lifecycle (config, bootstrap, discovery)

Service Authorization (RBAC)

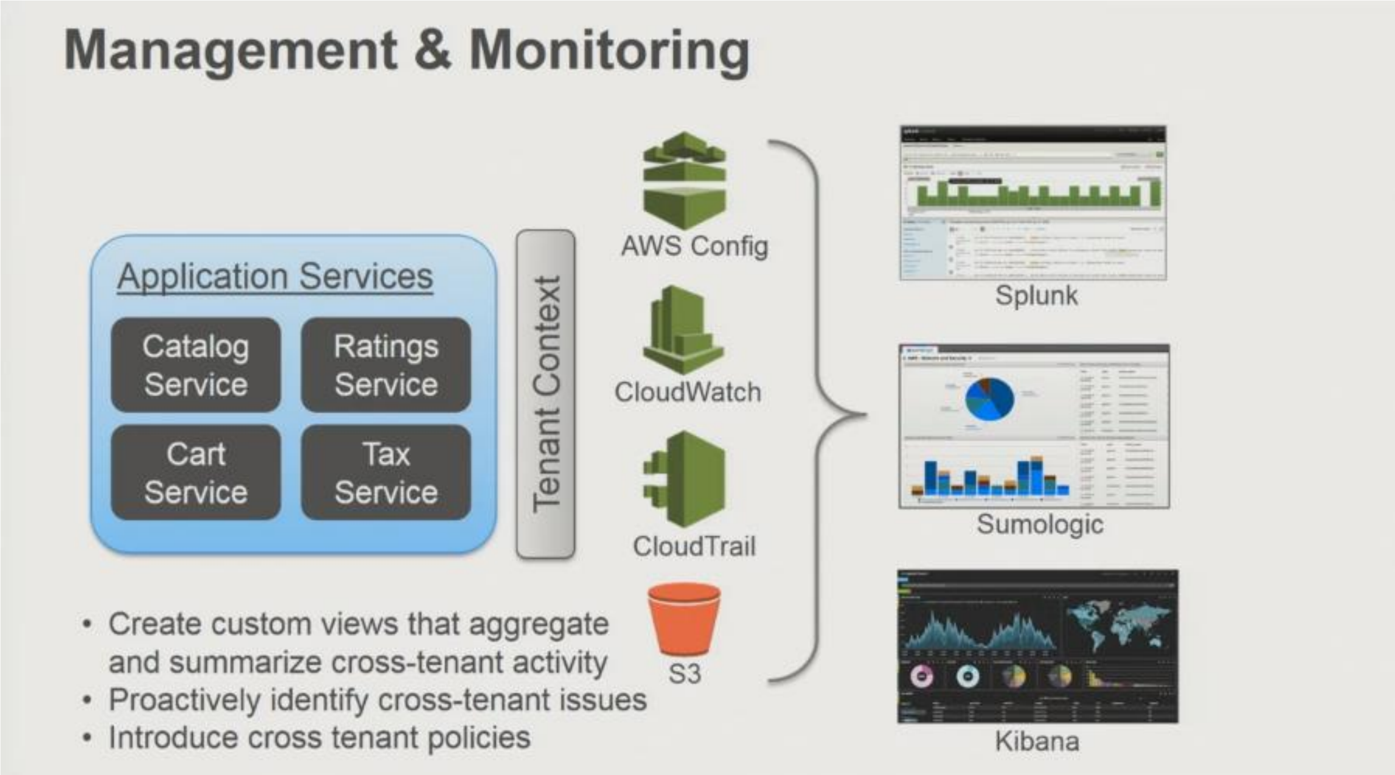
Your Service Code

Data Access

Logging

Analytics

Metrics



Management & Monitoring

Application Services

Catalog Service

Ratings Service

Cart Service

Tax Service

Tenant Context

AWS Config

CloudWatch

CloudTrail

S3

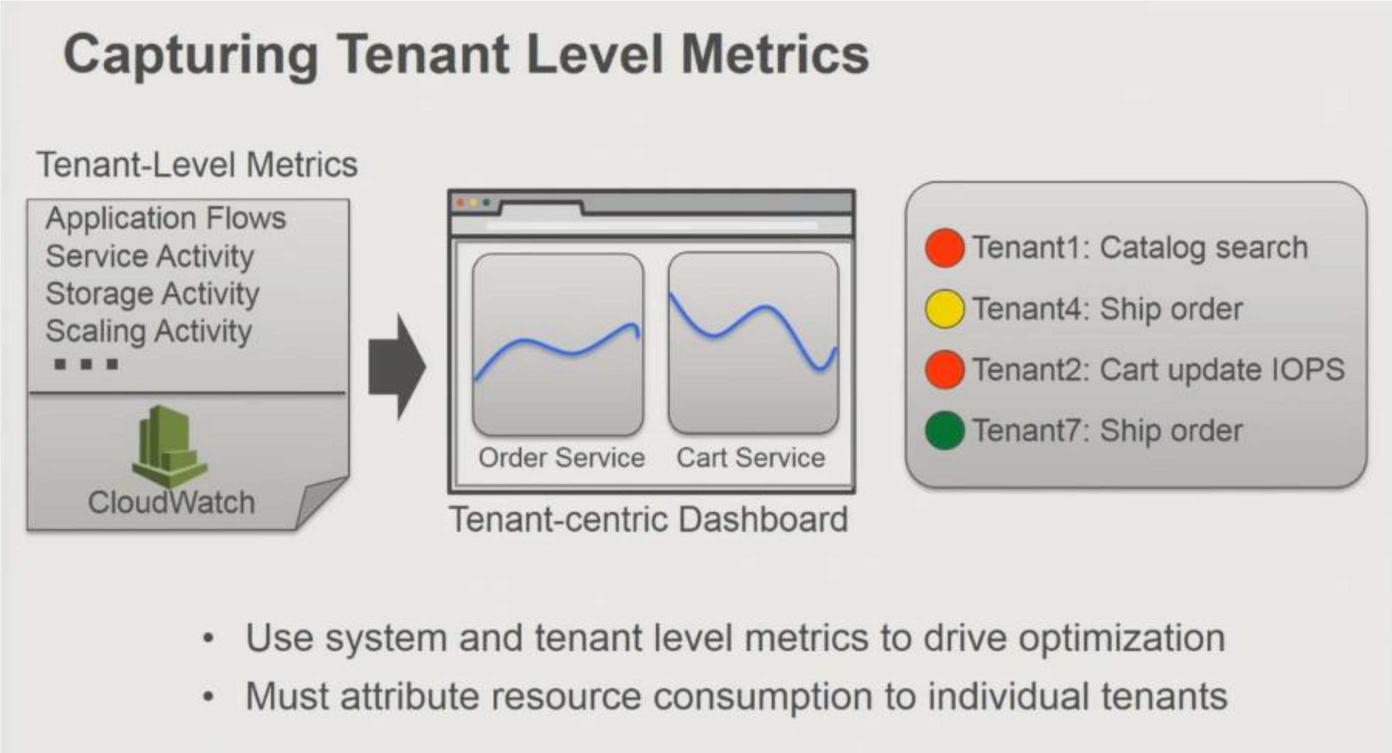
Splunk

Sumologic

Kibana

- Create custom views that aggregate and summarize cross-tenant activity
- Proactively identify cross-tenant issues
- Introduce cross tenant policies

We can use tools to aggregate logs, see trends and analytics. We still need more tools to help us see the cross-tenant view and at the multi-tenant level to see where tenants may be experiencing problems and how we can help resolve them.



Capturing Tenant Level Metrics

Tenant-Level Metrics

- Application Flows
- Service Activity
- Storage Activity
- Scaling Activity
- ■ ■ ■

CloudWatch

Order Service

Cart Service

Tenant-centric Dashboard

🔴 Tenant1: Catalog search

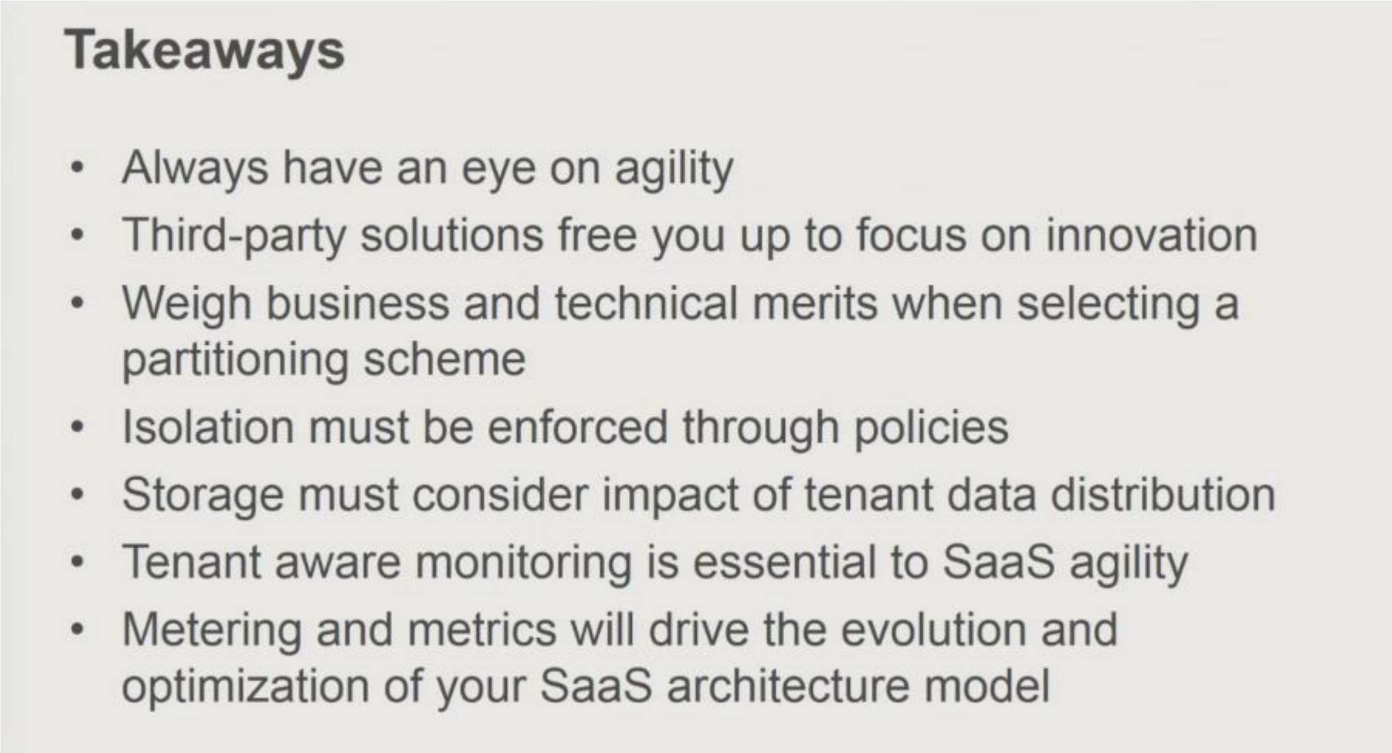
🟡 Tenant4: Ship order

🔴 Tenant2: Cart update IOPS

🟢 Tenant7: Ship order

- Use system and tenant level metrics to drive optimization
- Must attribute resource consumption to individual tenants

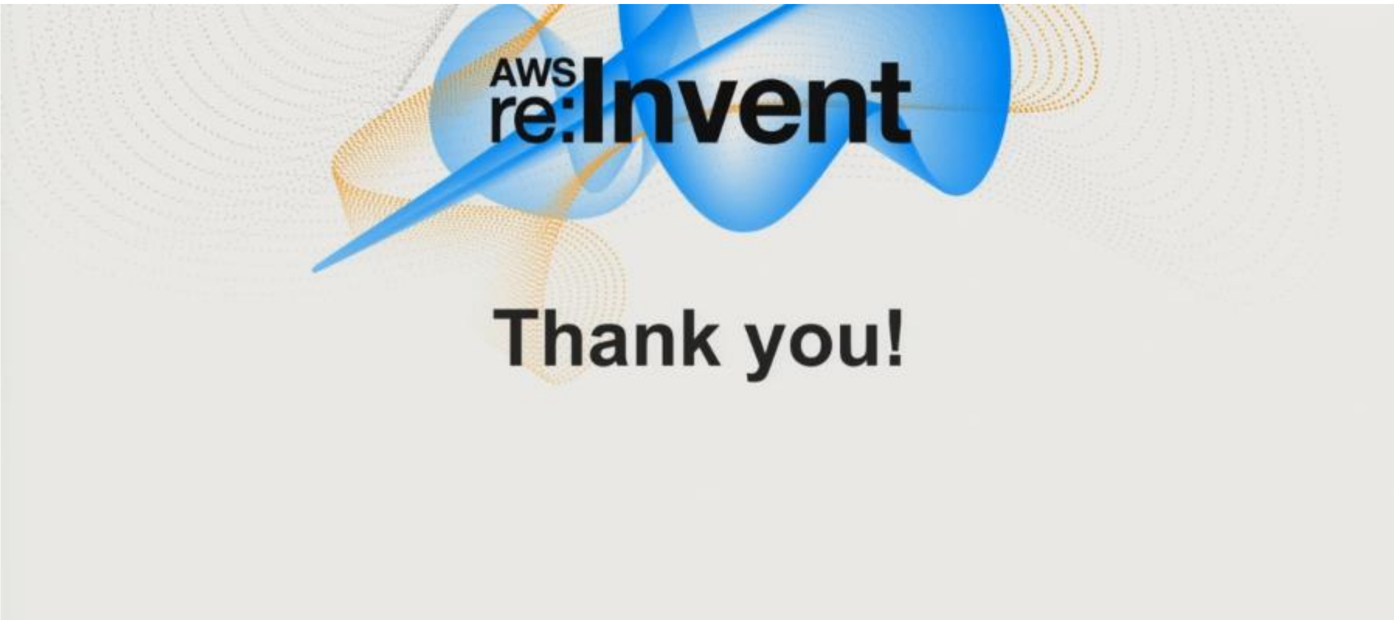
You might actually need to instrument the actual microservices within your platform for custom metrics that can be aggregated into views and dashboards



Takeaways

- Always have an eye on agility
- Third-party solutions free you up to focus on innovation
- Weigh business and technical merits when selecting a partitioning scheme
- Isolation must be enforced through policies
- Storage must consider impact of tenant data distribution
- Tenant aware monitoring is essential to SaaS agility
- Metering and metrics will drive the evolution and optimization of your SaaS architecture model

The goal of SaaS is to increase agility and the ability to add more features to the products quickly and be nimbler to respond to change in the customer expectations and competitive landscape. SaaS environments live and breathe based on the metrics they can collect and use.



AWS re:Invent

Thank you!