



Partner
Network

SaaS Enablement Framework


Tod Golding, Partner Solutions Architect
todg@amazon.com

Agenda

- SaaS on AWS
- Framework Overview
- Framework Components
- Guiding Principles

Why SaaS?

Customers of all sizes want their software delivered to them as SaaS

 Cloud software will grow to **\$112.8 billion by 2019** at a compound annual **growth rate of 18.3%** and will significantly outpace traditional software product delivery, **growing nearly five times faster** than the software market as a whole and becoming the significant growth driver to all functional software markets. ... **30% of all new business software purchases will be of service-enabled software.**”

Why SaaS on AWS?

SaaS characteristics:

- ✓ Priced on usage or users
- ✓ Self-service on-demand
- ✓ Multi-tenant, shared infrastructure
- ✓ Elastic usage

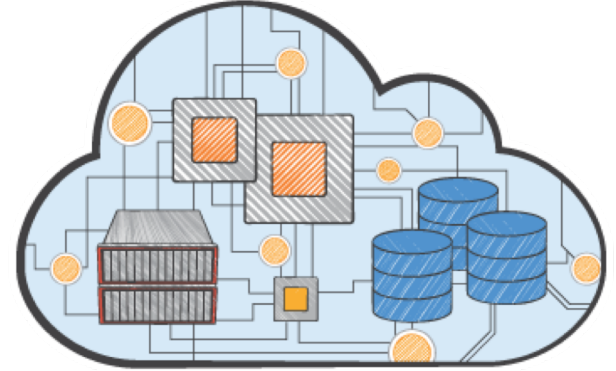
AWS characteristics:

- ✓ Pay only for what you use
- ✓ Resources on-demand
- ✓ Highly scalable and durable services
- ✓ Auto-scaled and scriptable resources

SaaS != hosted application management

SaaS on AWS lets you focus on your customers

1. AWS flexibility → Quickly respond to changes in application needs in a cost effective way
2. AWS innovation → Keep up with latest industry trends without building it all yourself
3. AWS ecosystem → Leverage the experience and knowledge of others on the AWS platform



Framework Overview

A review of the key goals and elements of the SaaS Enablement Framework.

Framework Goals

- Provide a roadmap for the development, operation, and launch of SaaS offerings on AWS
- Classify SaaS best practices and design tradeoffs
- Be a catalyst for SaaS community solution development and discovery
- Streamline and accelerate development of SaaS solutions

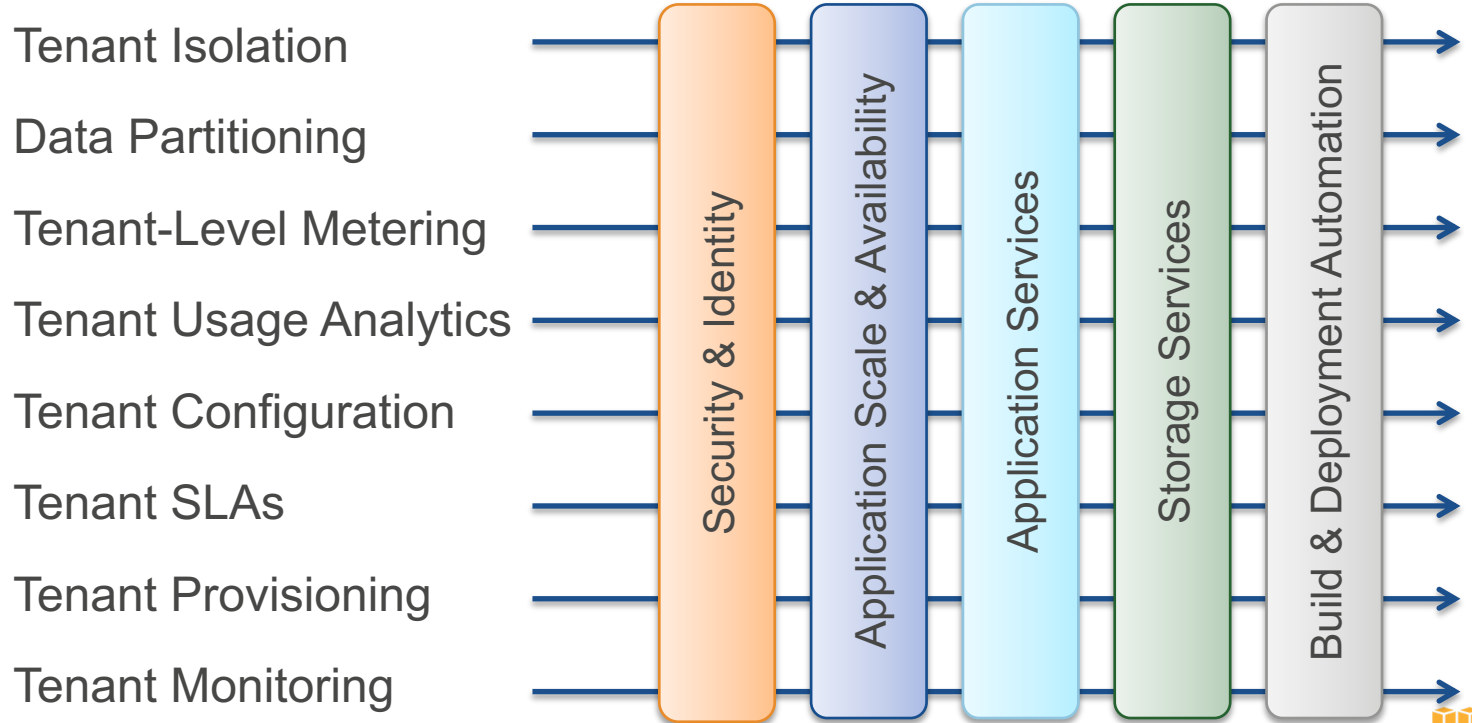
Framework Scope

A collection of core concepts and practices that guide the delivery, design, and operation of SaaS solutions on AWS

- Access
- Build
- Manage
- Profile
- Optimize
- Migrate/Transform
- Sell

Where SaaS Fits

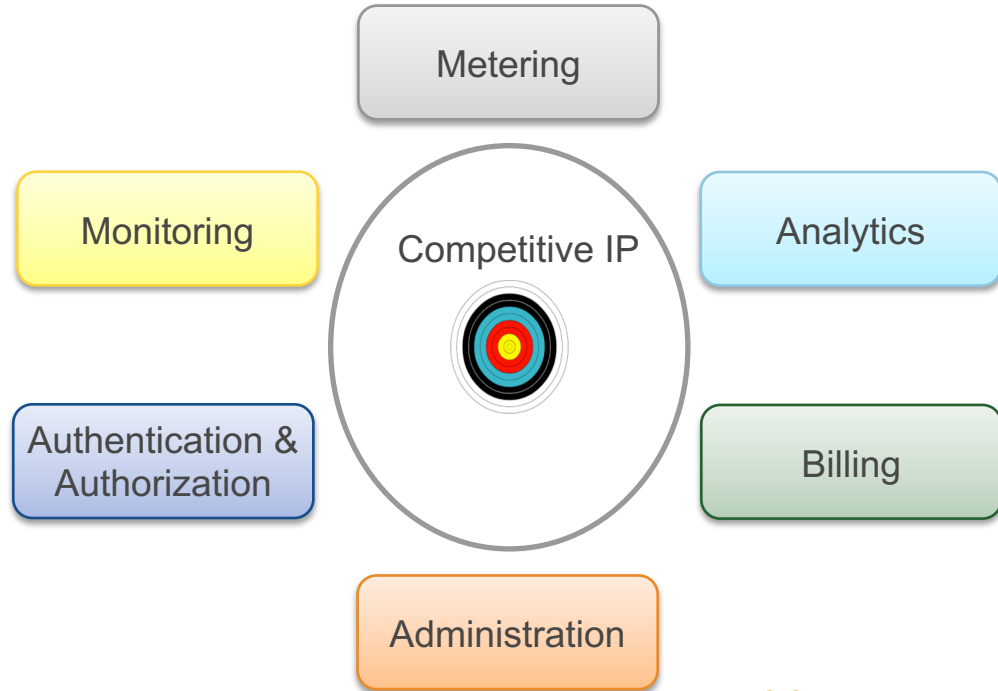
Tenancy and business considerations introduce a layer of SaaS specific best practices that touch most dimensions of the AWS stack of services



Common Solutions to Common Problems

Provide examples of custom and/or provider-supported integrations that can accelerate SaaS adoption

- Enumerates and defines a fundamental set of SaaS core services
- Establishes roles and boundaries for each service
- Provides examples of custom and/or partner-supported service integrations
- Promotes a model for ecosystem extension and collaboration
- Put the focus where it should be—on competitive IP



Leveraging APN Partner Solutions

Providing tenant aware integrations of provider tools that address core functionality

Monitoring



Authentication/Authorization



Analytics



Metering



Billing



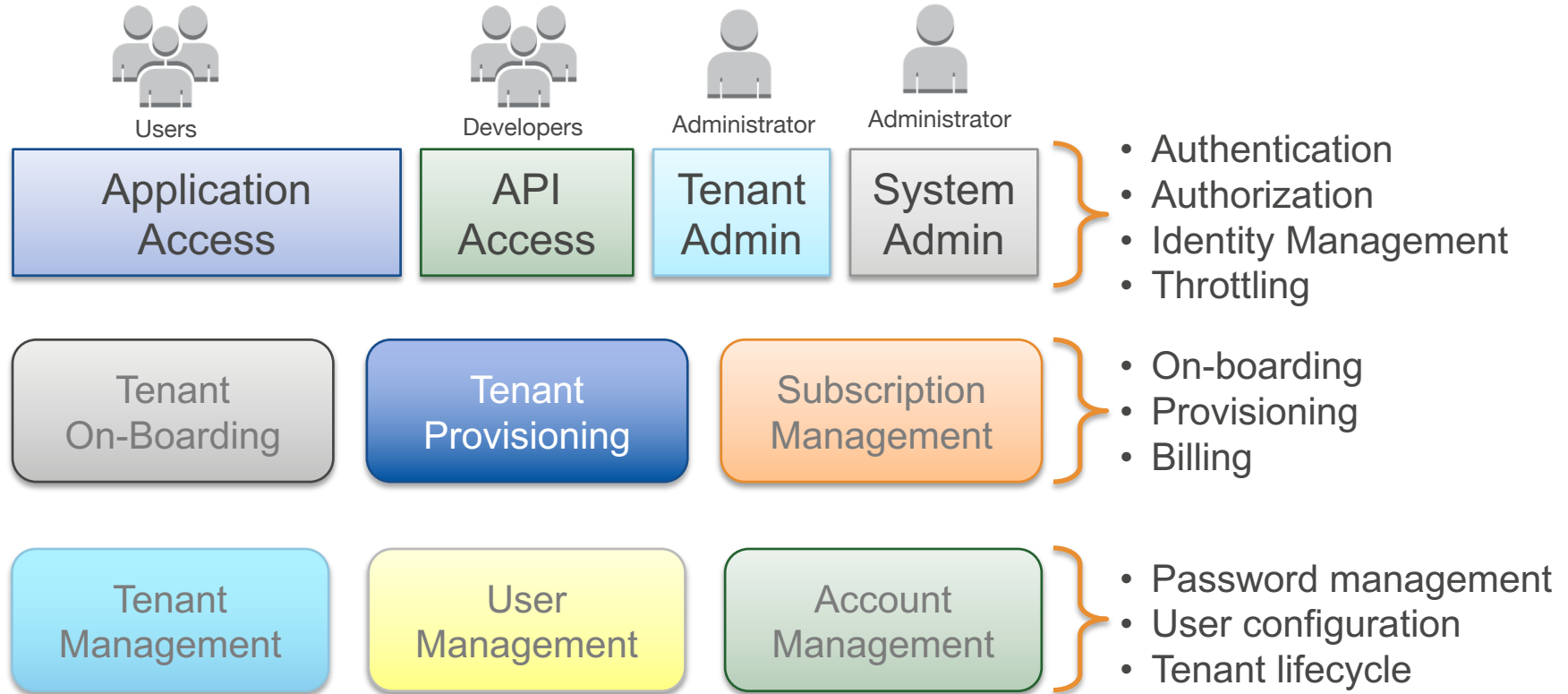
Key Framework Tenets

- Don't separate business from the technology
- Place a premium on analytics and metering
- Profile your tenant's security expectations
 - Understand your near and long-term isolation needs
- Model compliance requirements early
- Make operational efficiency and visibility a priority
- Make agility a priority

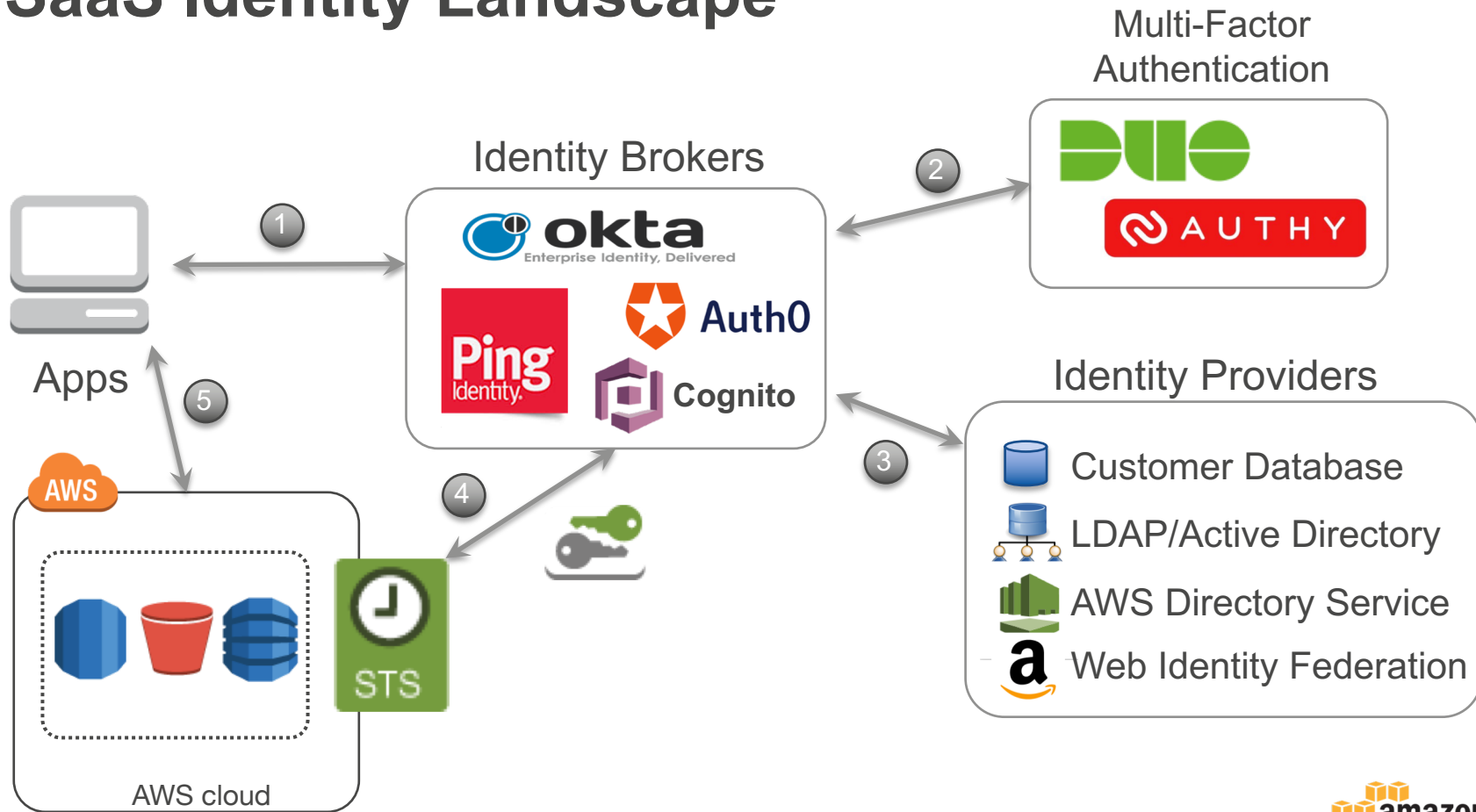
Access

Key considerations for managing and controlling access to SaaS solutions

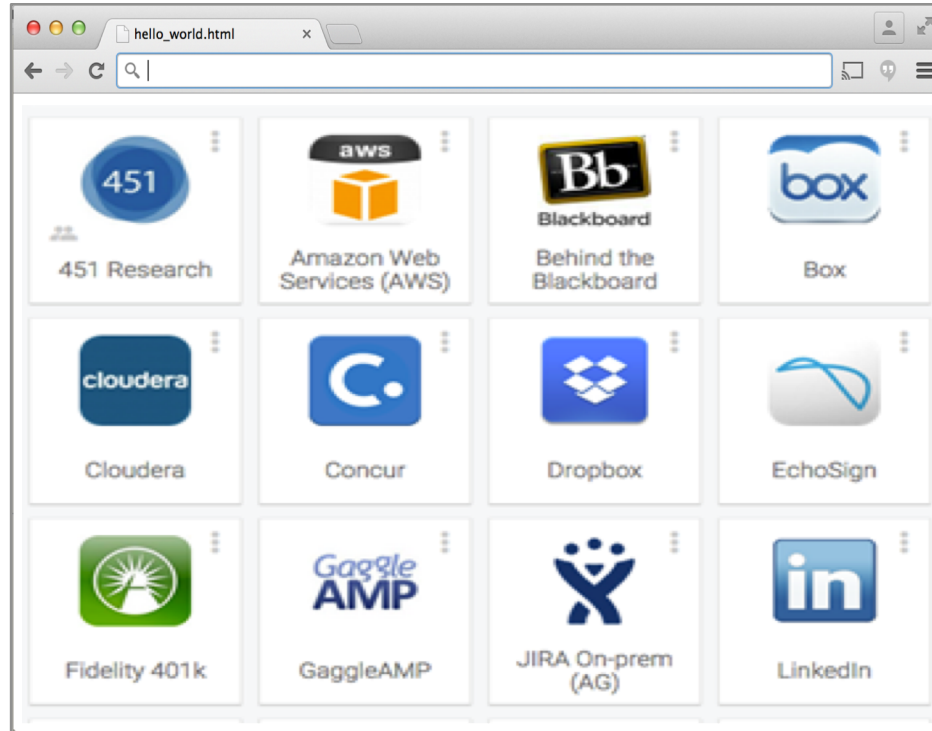
Managing the Front Door



SaaS Identity Landscape



Centralized SSO On-Boarding



SaaS Application Dashboard

Identity Design Strategies

- Identity should be a day one consideration
- Limit the number of identities a user needs to manage
- Let the IdPs own overhead and risk of storing identity and managing authentication
- Leverage identity brokers to decouple from growing number of IdPs
- Store application-specific attributes in a database linked to a user
- RBAC applied to all layers of access

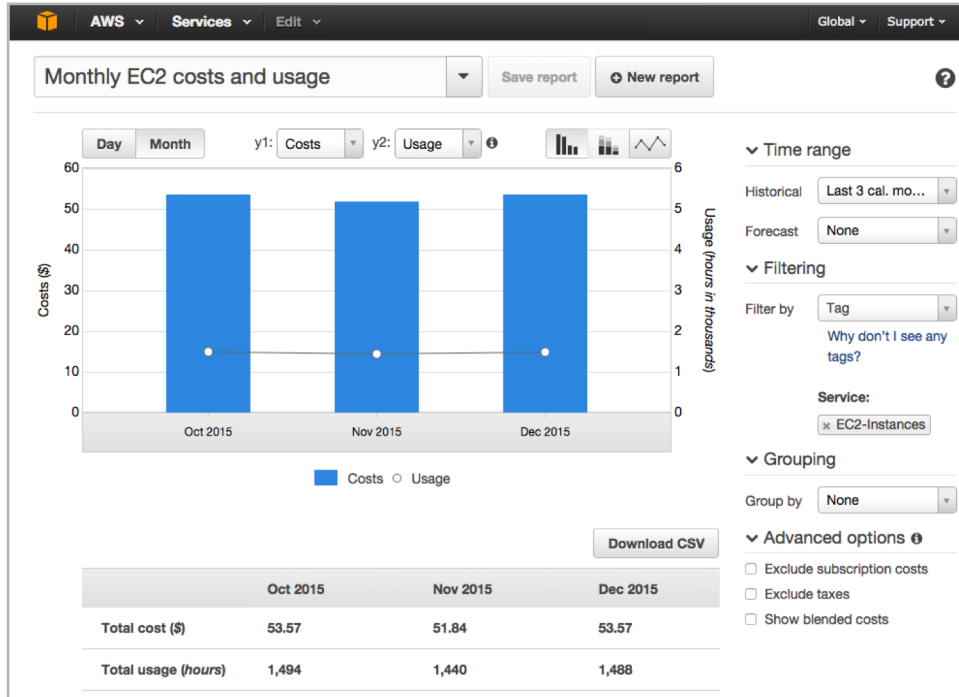
API Design Considerations

- Minimize cross-tenant impacts
- Leveraging API throttling
 - A single tenant should never disproportionately degrade performance
- Building in fault tolerance
 - Applying tenant circuit breakers
 - Leveraging asynchronous messaging
 - Supporting partial outages
- Tenant specific SLAs

Subscription Management (Billing)

- Developing a tiered pricing model
 - Correlating pricing with cost footprint
 - Common patterns for modeling pricing
- Leveraging metering data
 - Using meter data to control resource utilization
 - Putting hard caps on consumption
 - Promoting and facilitating tier upgrades
- Integrating with provider billing solutions
 - Configuring and enforcing billing plans
 - Integrating with partner billing solutions

Provisioning Tags for Billing

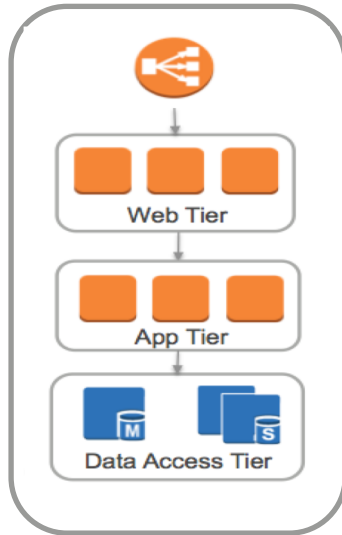


- Assign tags to AWS resources during provisioning
- Leverage tags as model for aggregation tenant load
- Enables use of cost allocation reports

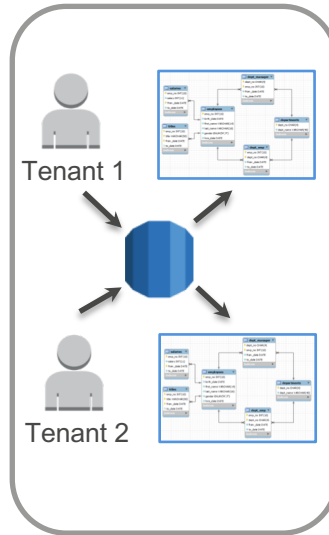
Build

A review of the common patterns and practices that are used to design, architect, deploy, and validate SaaS solutions on AWS

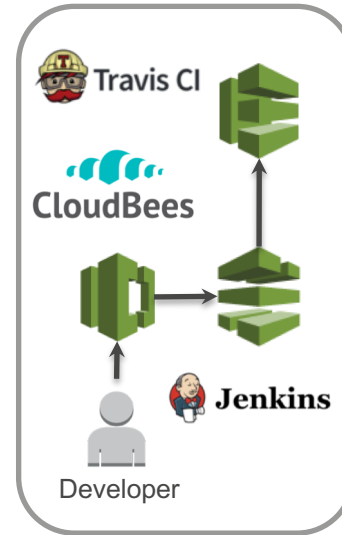
Build Overview



Design



Partition



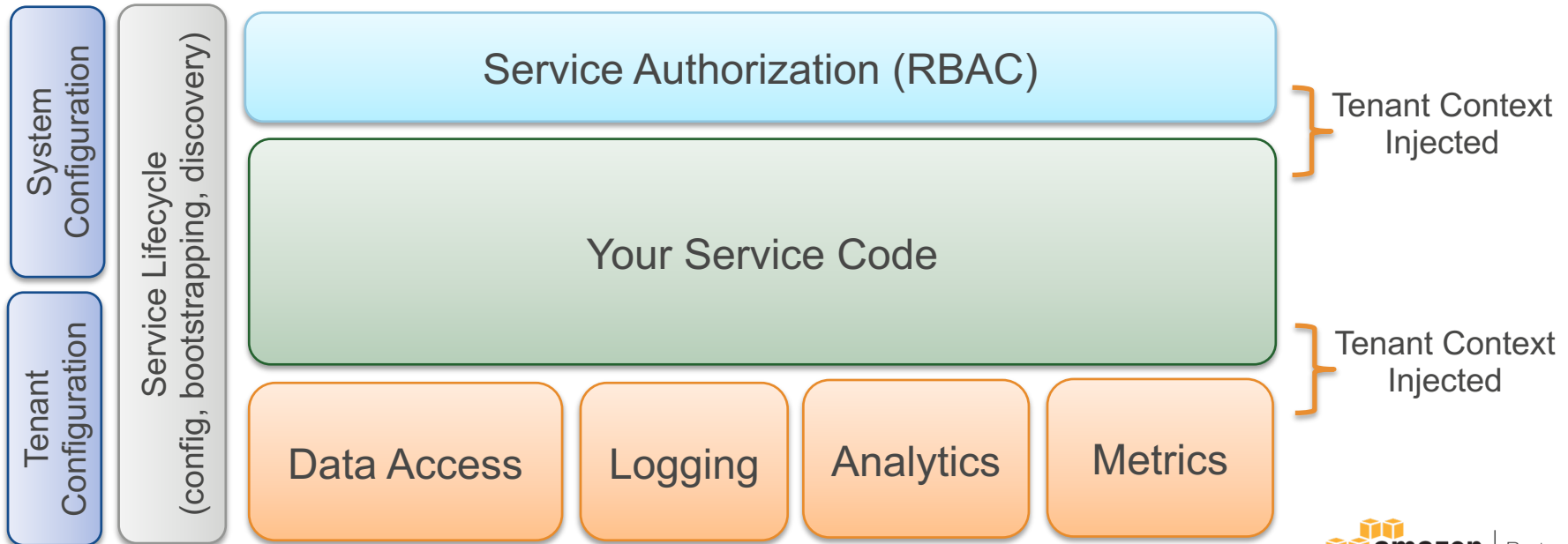
Deploy

Design and Tenant Profiling

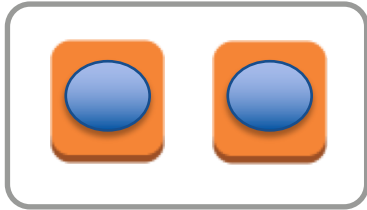
- SaaS design must be aligned with tenant profile
- Assess and capture tenant requirements
 - What are the security requirements of your tenants?
 - What levels of isolation are relevant to your tenants?
 - Do your tenants need to meet specific compliance criteria?
 - What is the profile of typical tenant load (spikey, flat, etc.)
 - What kind of SLA's are expected of your application?
 - How much customization is needed for each tenant?
 - What features will distinguish each tier of your SaaS offering?

Designing Multi-Tenant Services

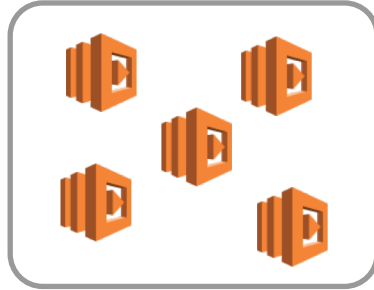
- Abstracting away tenant awareness & policies
- Maximizing developer productivity
- Centrally managed policies/configuration



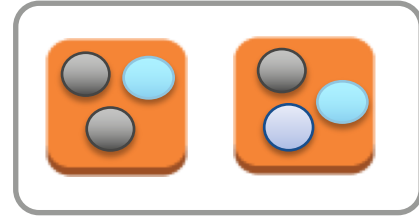
Service Scaling & Decomposition Strategies



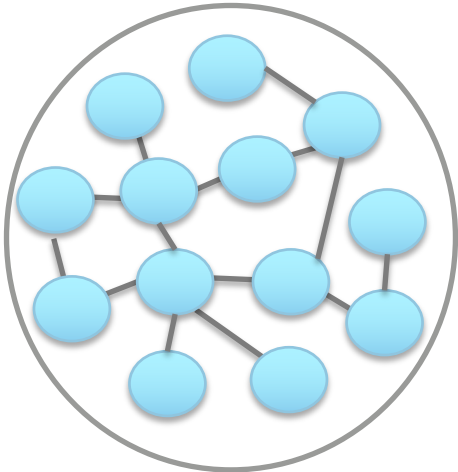
Amazon EC2 Instances



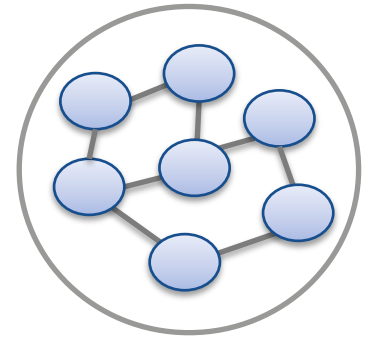
AWS Lambda Functions



Amazon ECS Container Cluster

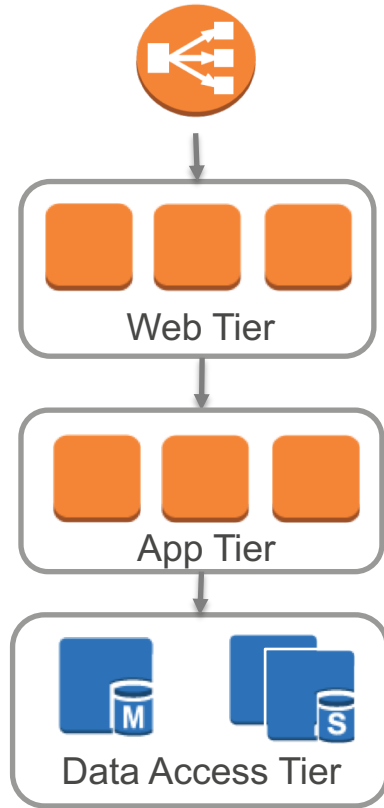


- Higher density of services improves SaaS agility and availability
- More services creates more granularity for tuning elasticity & tenant experience
- Service density adds complexity to deployment
- Cost considerations influence model



Application Design Model: Variation 1

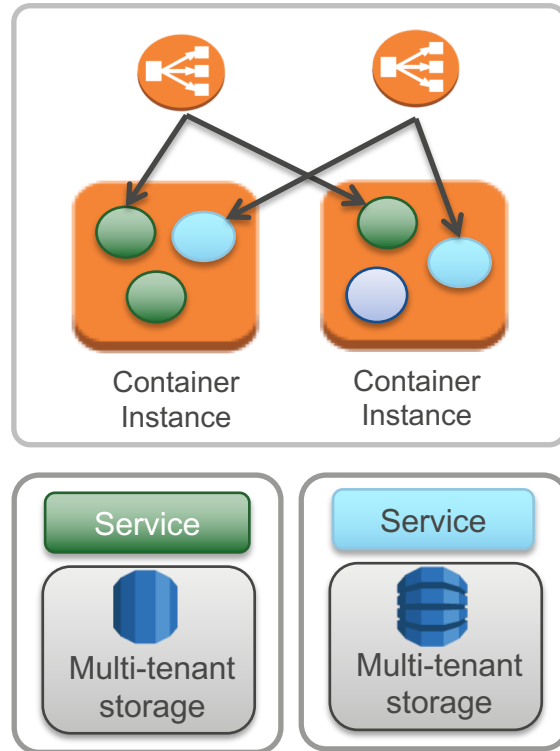
N-Tier Web Application Framework (single or multi-tenant)



- Aligns with n-tier frameworks
- Scaling is coarse-grained
- Tier = unit of scaled
- Limited ability to design for failures
- Limited ability to tune tenant experience
- Single, shared data representation
- Common migration pattern

Application Design Model: Variation 2

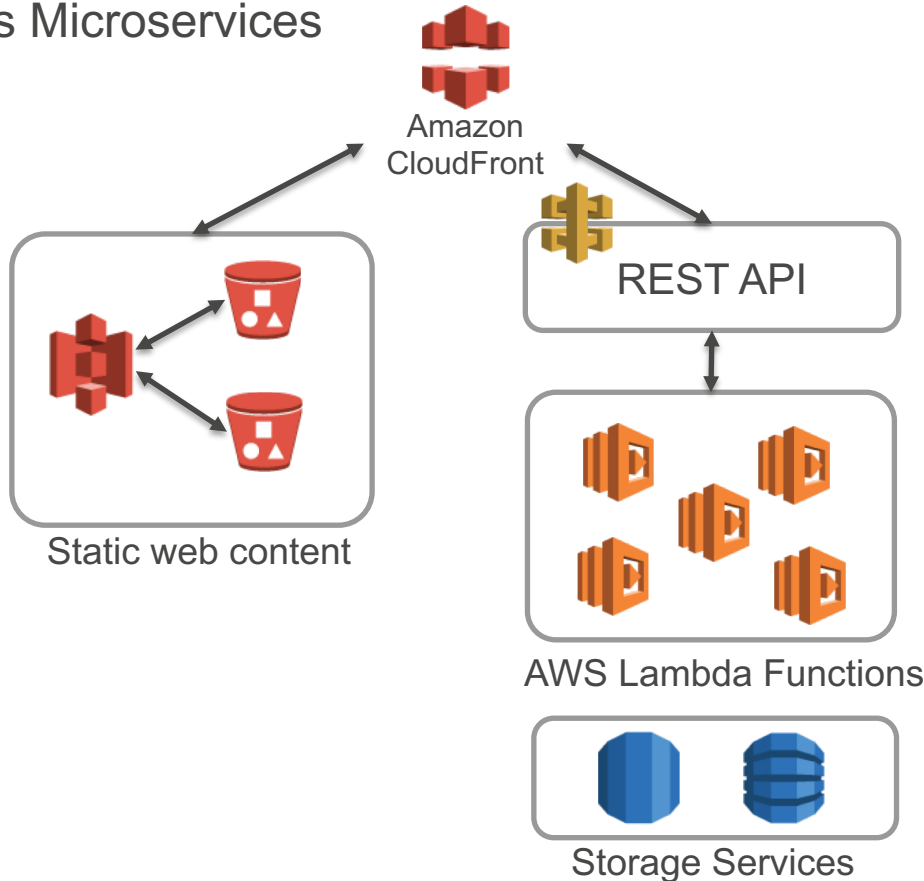
Containerized Microservices



- Services run in containers
- Cluster scales dynamically based on service and tenant load
- Each service is autonomous and owns its own storage model and multi-tenancy strategy
- Finer grained services
- More compelling isolation models
- More custom tenant experience

Application Design Model: Variation 3

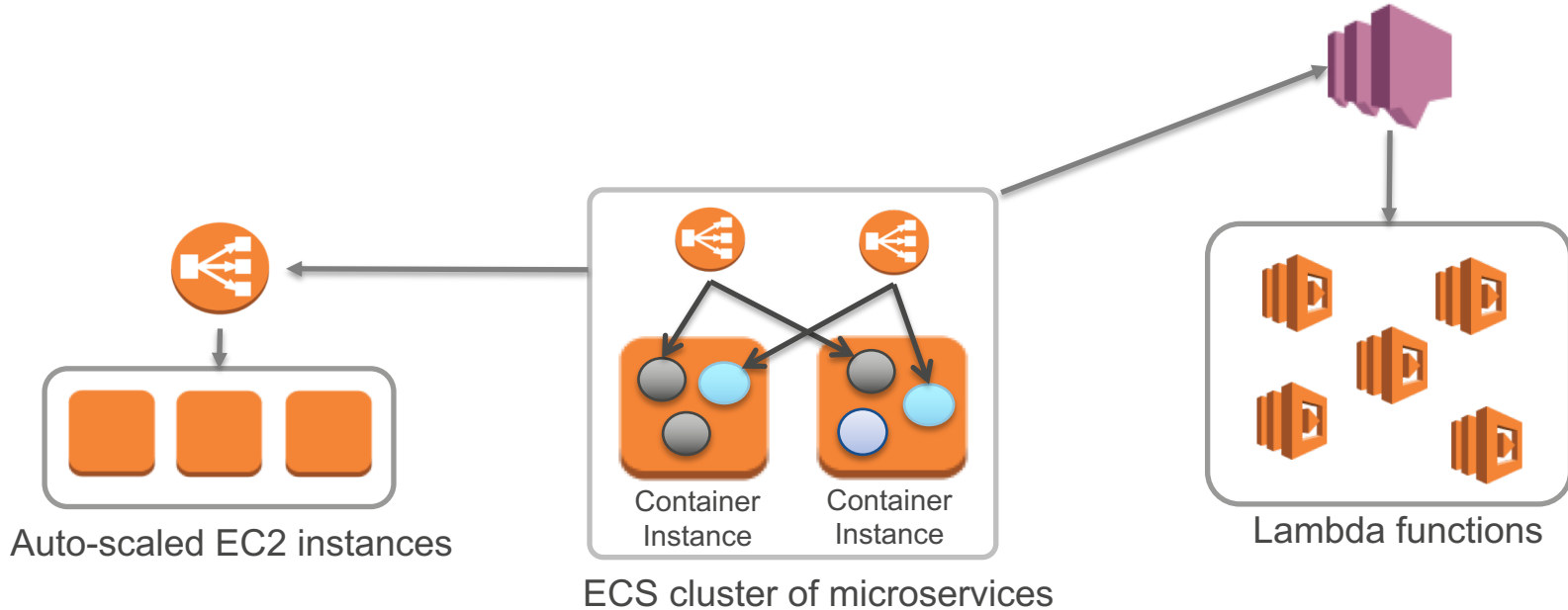
Serverless Microservices



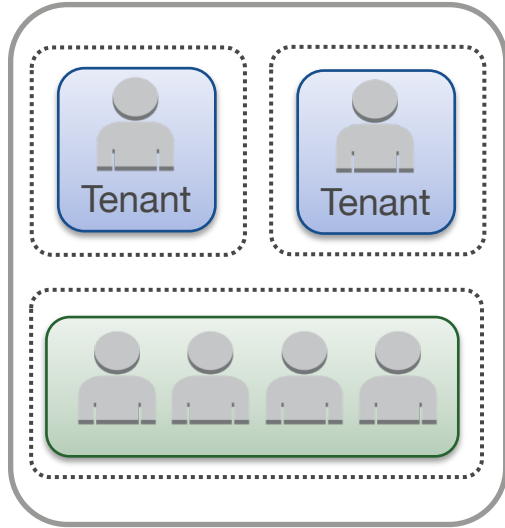
- Authentication/authorization
- Throttling managed by API
- Metering data captured
- Each function executes in a tenant context
- Serverless model optimizes spend a minimizes management footprint
- New models needs for profiling and assessing system health
- Trigger Lambda function from other Lambda functions
- Programming model changes
- Managing failures

Application Design Model: Variation 4

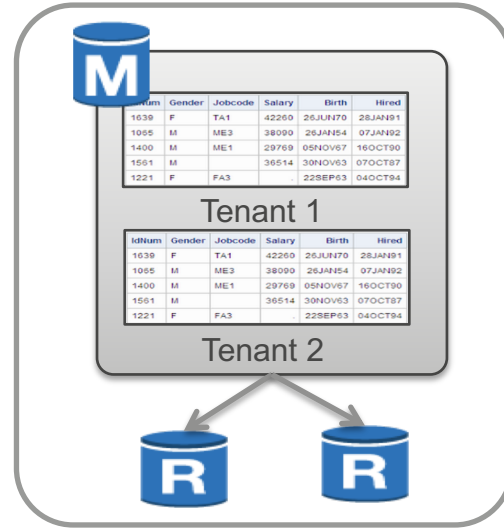
Hybrid of EC2, ECS, and Lambda scaling



Partition

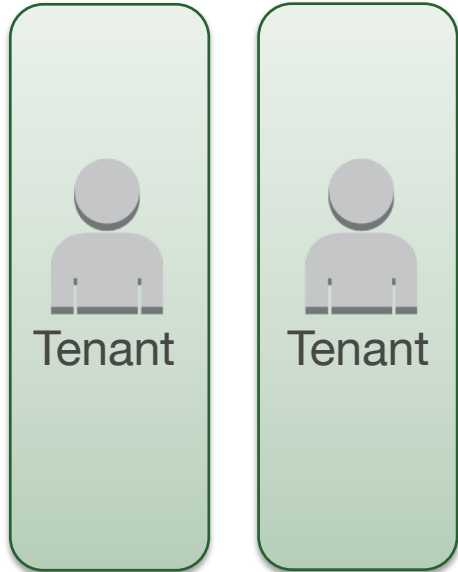


Tenant Partitioning

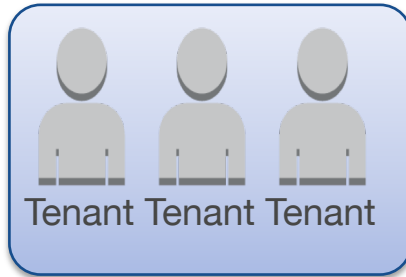
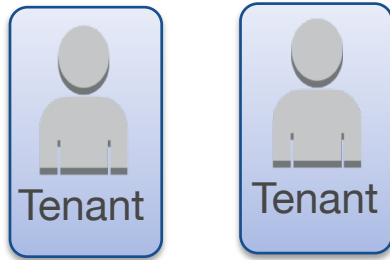


Data Partitioning

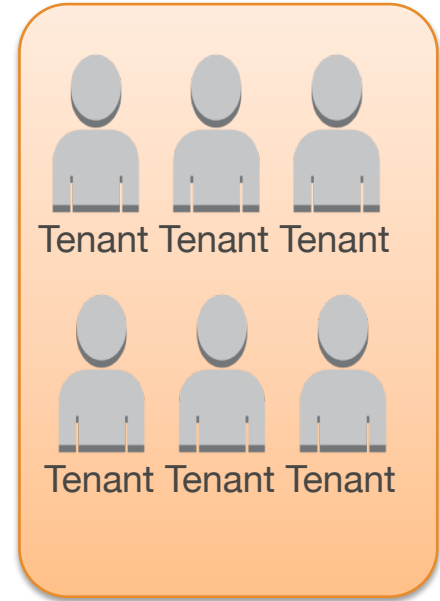
Partitioning Patterns



Silo



Bridge



Pool

Partitioning Spectrum

Silo Model

Bridge Model

Pool Model



Pros

- Security
- Dedicated infrastructure
- No cross-tenant impacts
- Tenant-specific tuning
- Tenant level availability

Cons

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

Pros

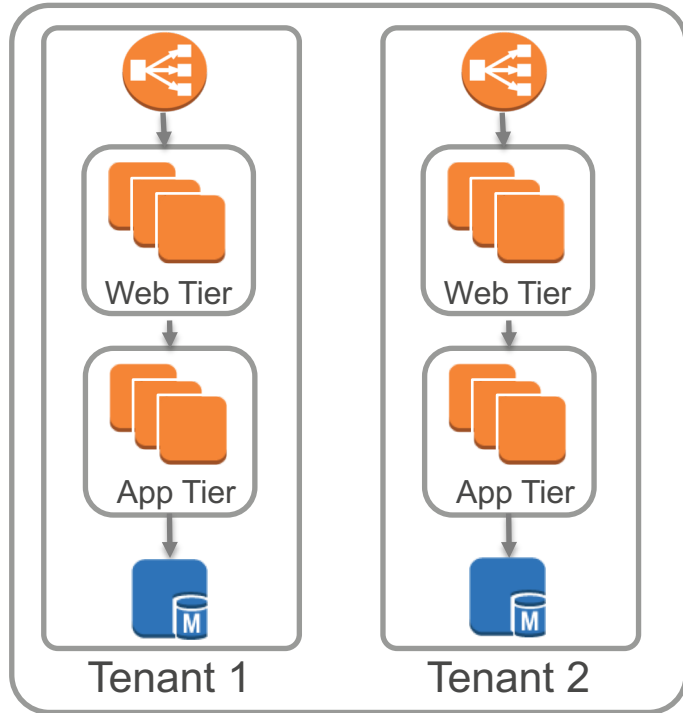
- Agility
- Cost optimization
- Centralized management
- Simplified deployment
- Analytics/metering aggregation

Cons

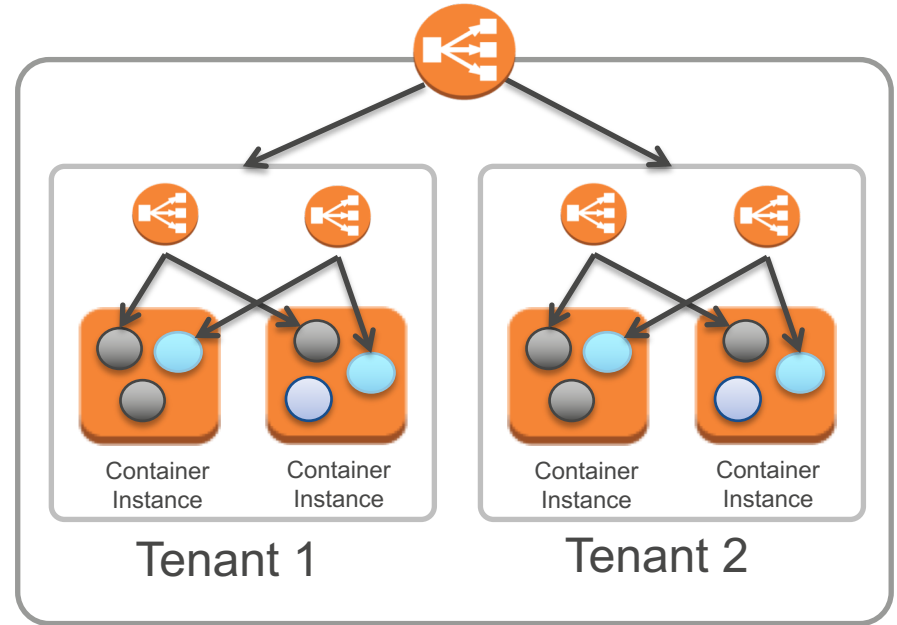
- Cross-tenant impacts
- Compliance challenges
- All or nothing availability

Tenant Isolation Models

Each application design scheme can be deployed in a single tenant model:



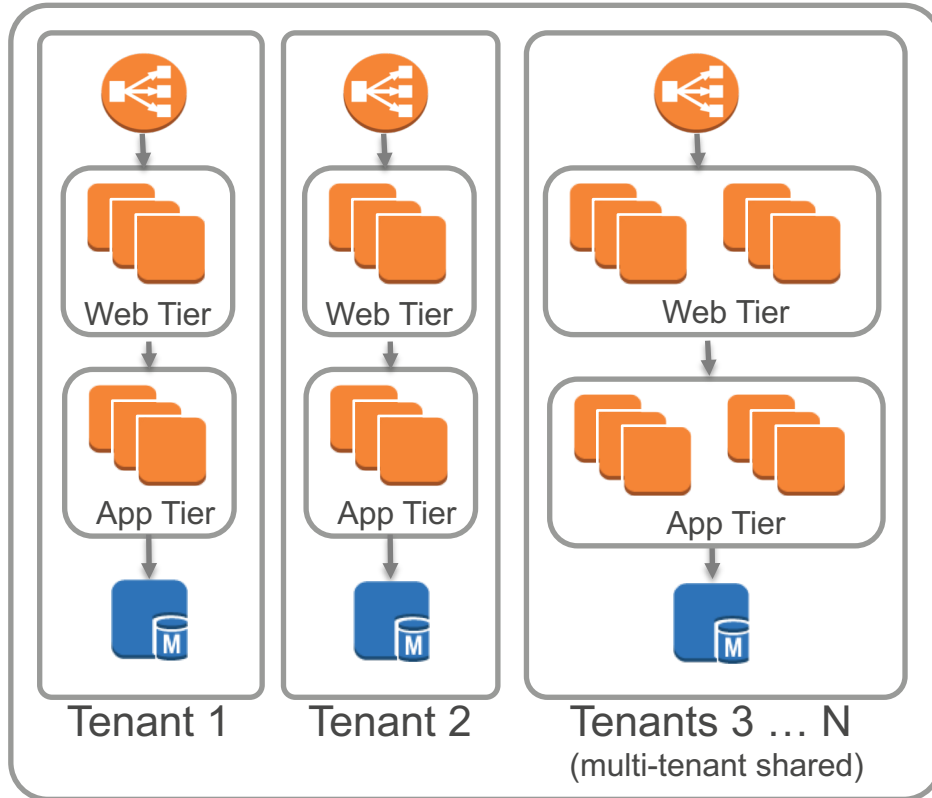
Full Stack Isolation



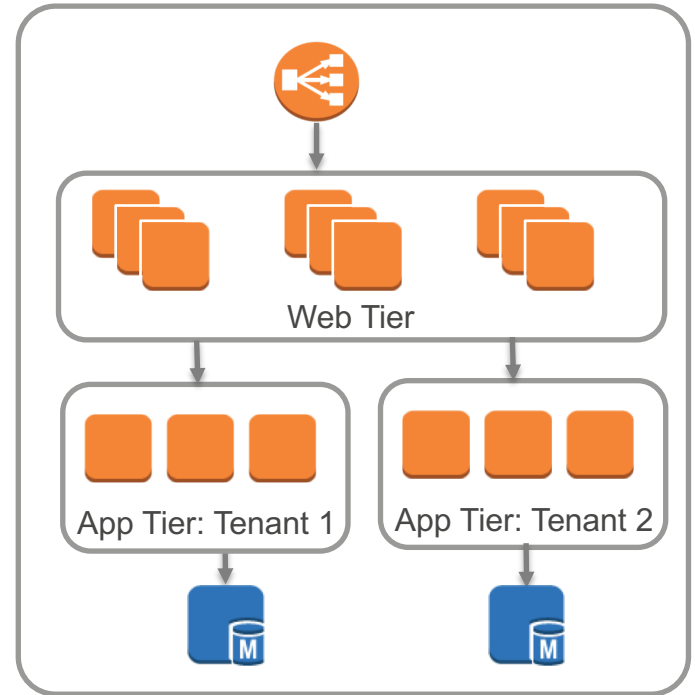
Container Isolation

Hybrid Isolation Model

Mix of single and multi-tenant models

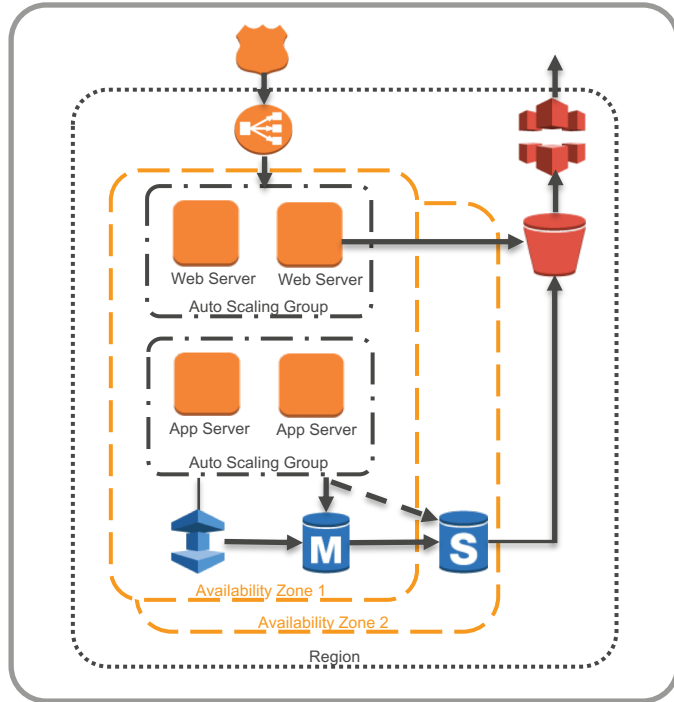


Single Tenant Application Layer

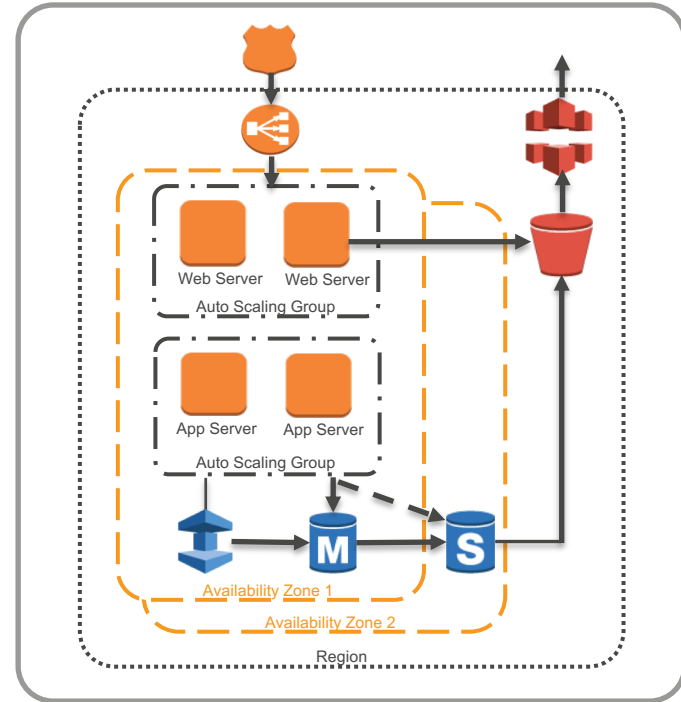


Account-Based Tenant Isolation

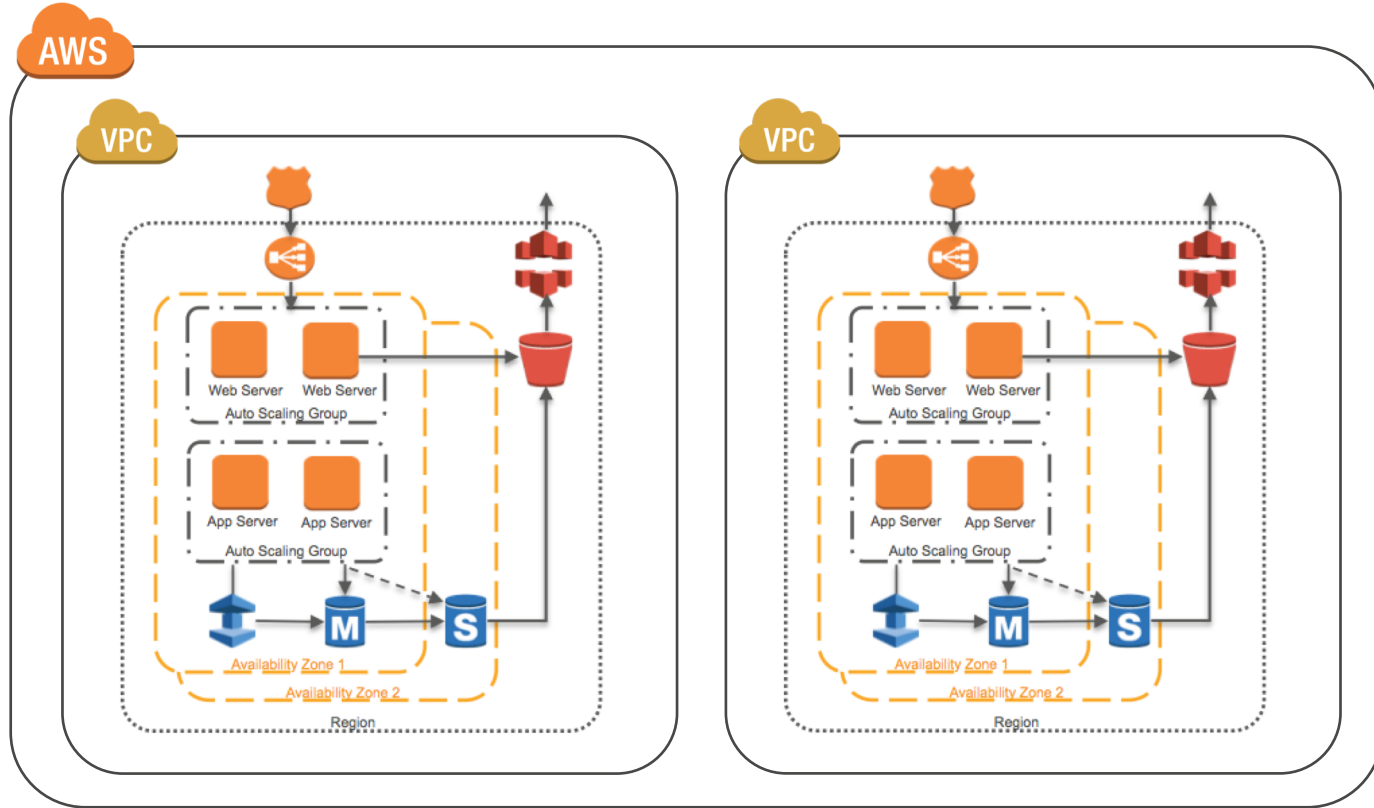
Tenant 1 (AWS Account A)



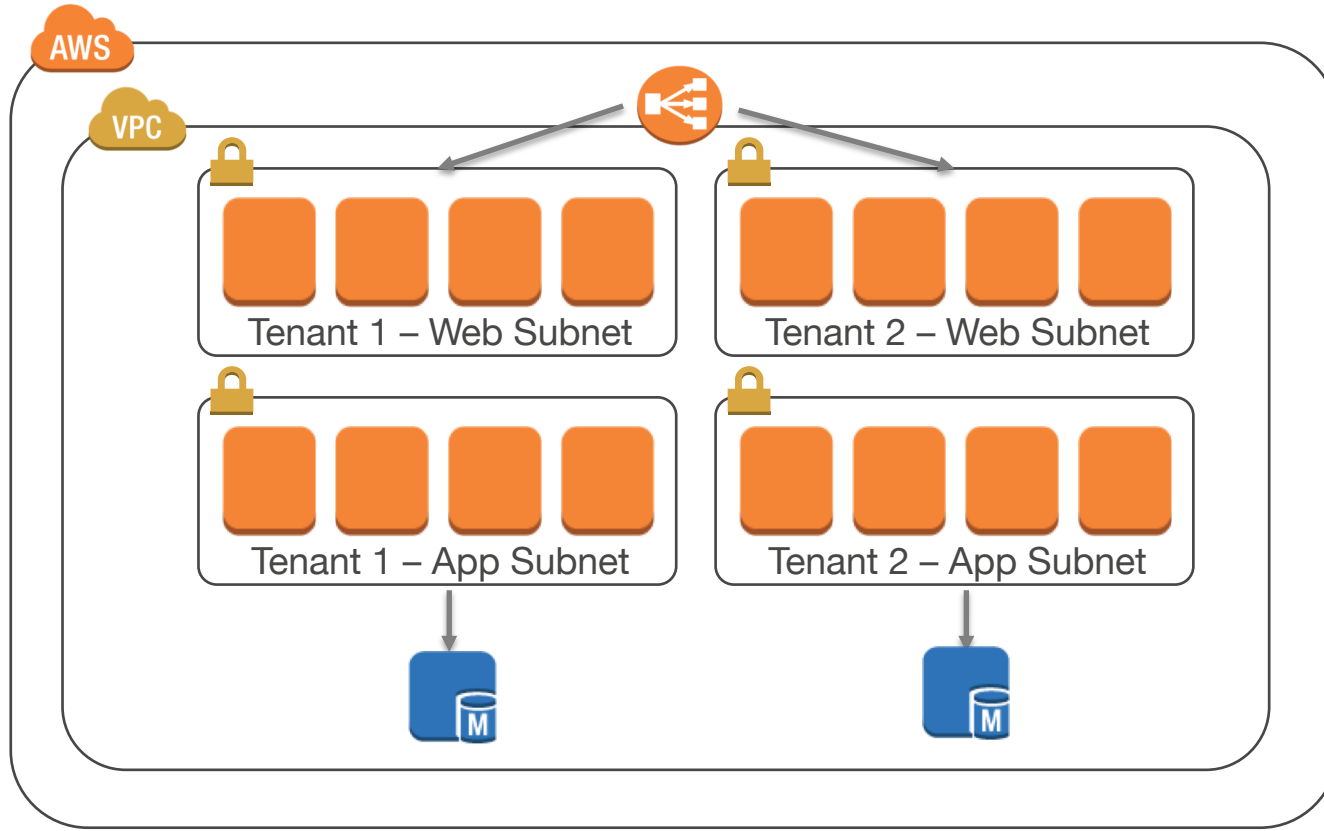
Tenant 2 (AWS Account B)



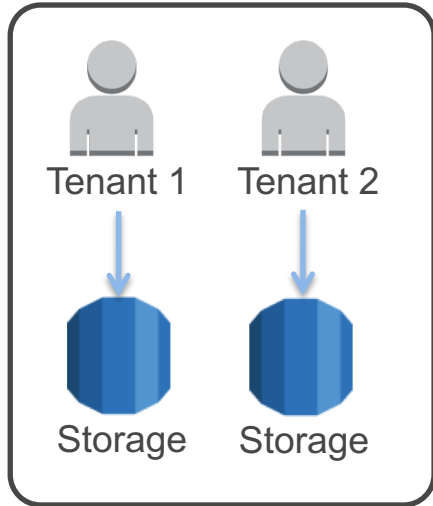
VPC-Based Tenant Isolation



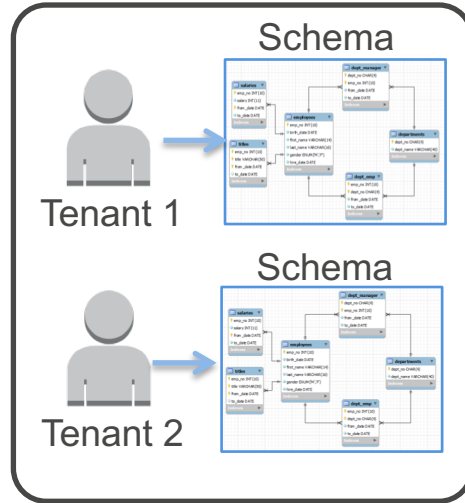
Subnet-Based Tenant Isolation



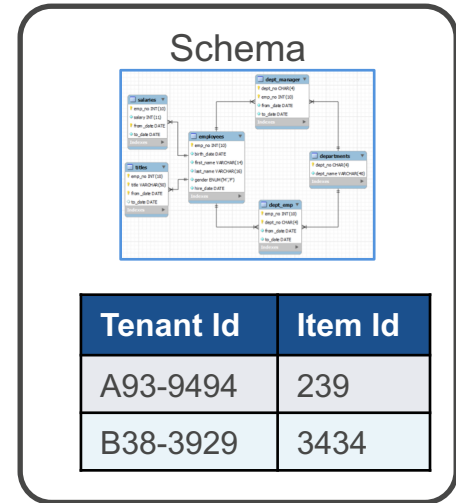
Data Partitioning Models



Separate database
for each tenant



Single database,
multiple schemas



Shared database,
single schema

Amazon RDS Data Partitioning Models

Instance Per Tenant

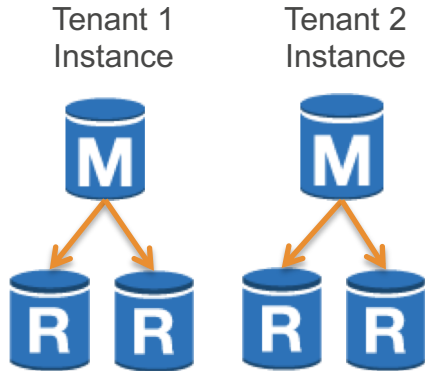
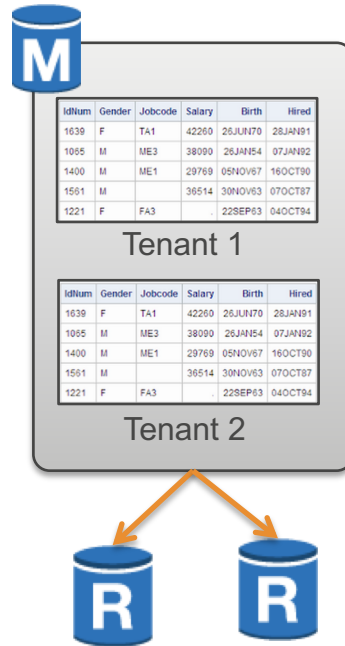
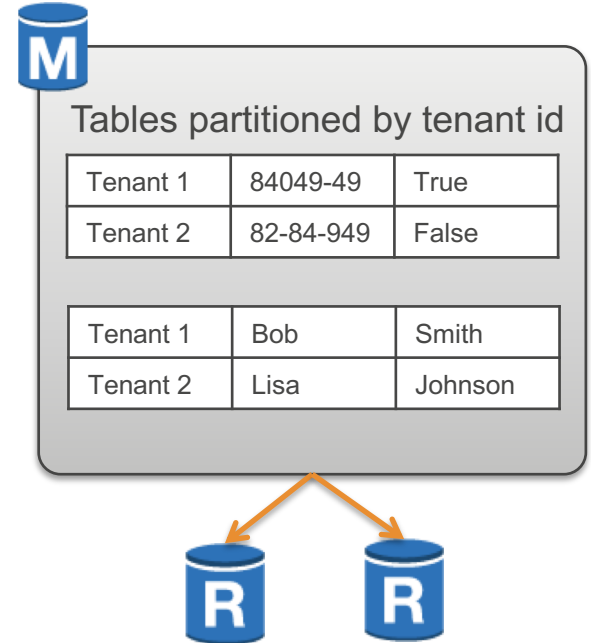


Table Per Tenant

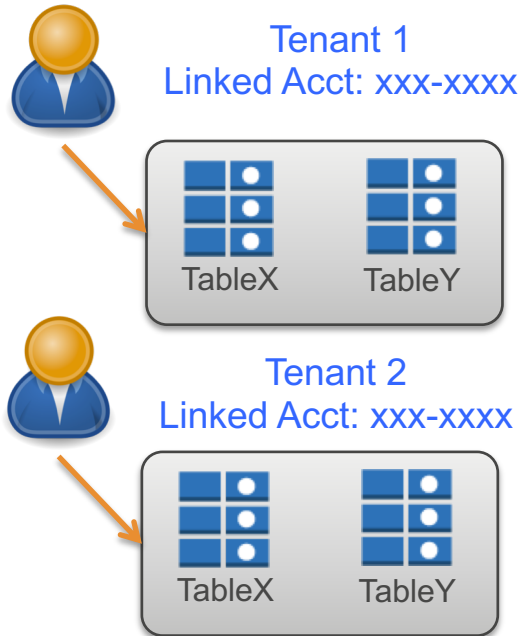


Multi-Tenant Tables

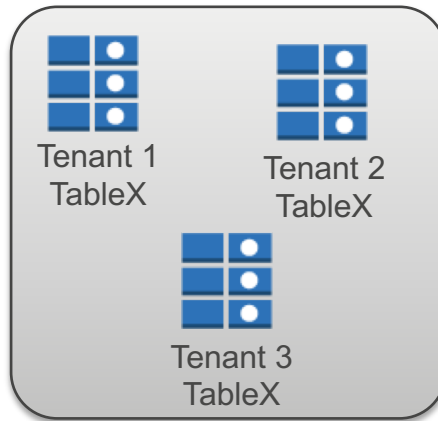


Amazon DynamoDB Data Partitioning Models

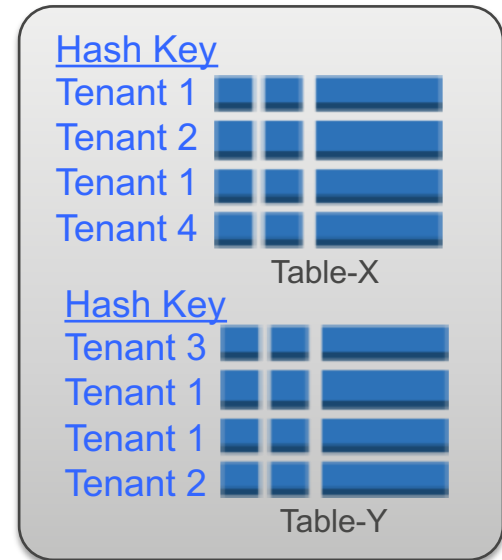
Linked Account Per Tenant



Tenant-Based Table Names



Tenant Indexed Tables



Partitioning & AWS Tools



AWS Identity & Access
Management



CloudWatch
Metrics



- Partitioning influences granularity of access, metrics, and control
- Control access to tenant instances and tables
- Capture tenant level metrics
- Manage and monitor storage resources



Amazon
RDS



Amazon
DynamoDB

Storage Scale & Operational Footprint

- Minimizing operational costs and complexity
- AWS storage services address key needs of SaaS solutions
 - Built-in mechanisms for safe-guarding high availability
 - Granular ability to tune IOPS
 - Ability to configure data backups and snap shots
 - Support for a variety of partitioning schemes
- A wide spectrum of storage services to address different solution profiles
 - Ability to balance cost profile with storage demands

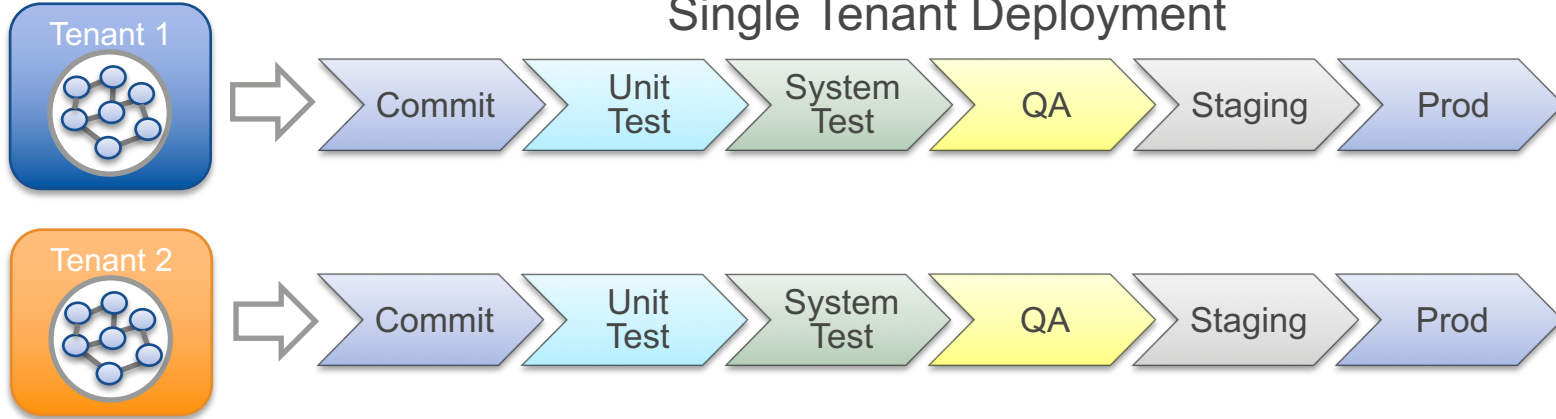
Deploy

- Deployment automation is the engine of SaaS agility
- View this as your competitive advantage
- Code and data must evolve in real-time
- Expect an ongoing investment in tooling and automation

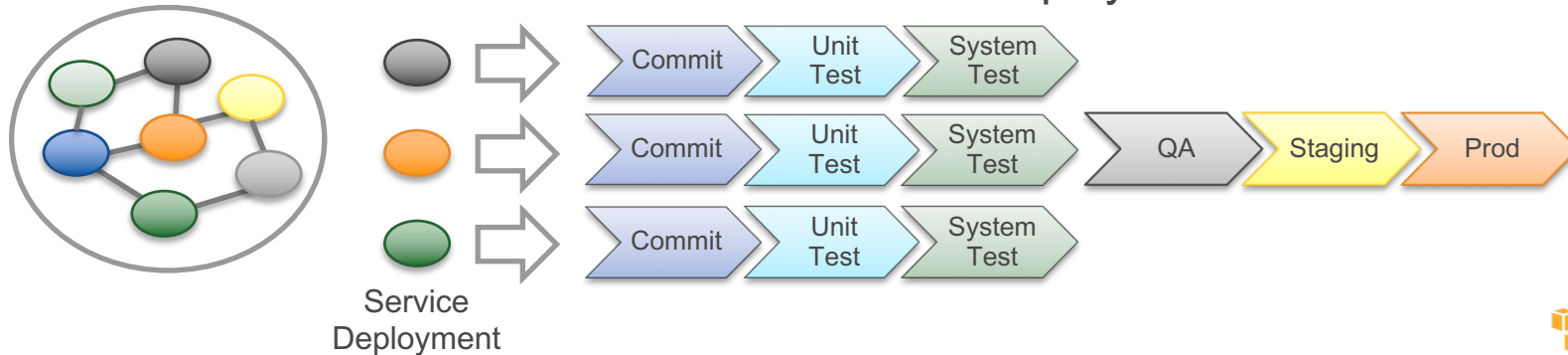


Tenant Deployment Models

Single Tenant Deployment



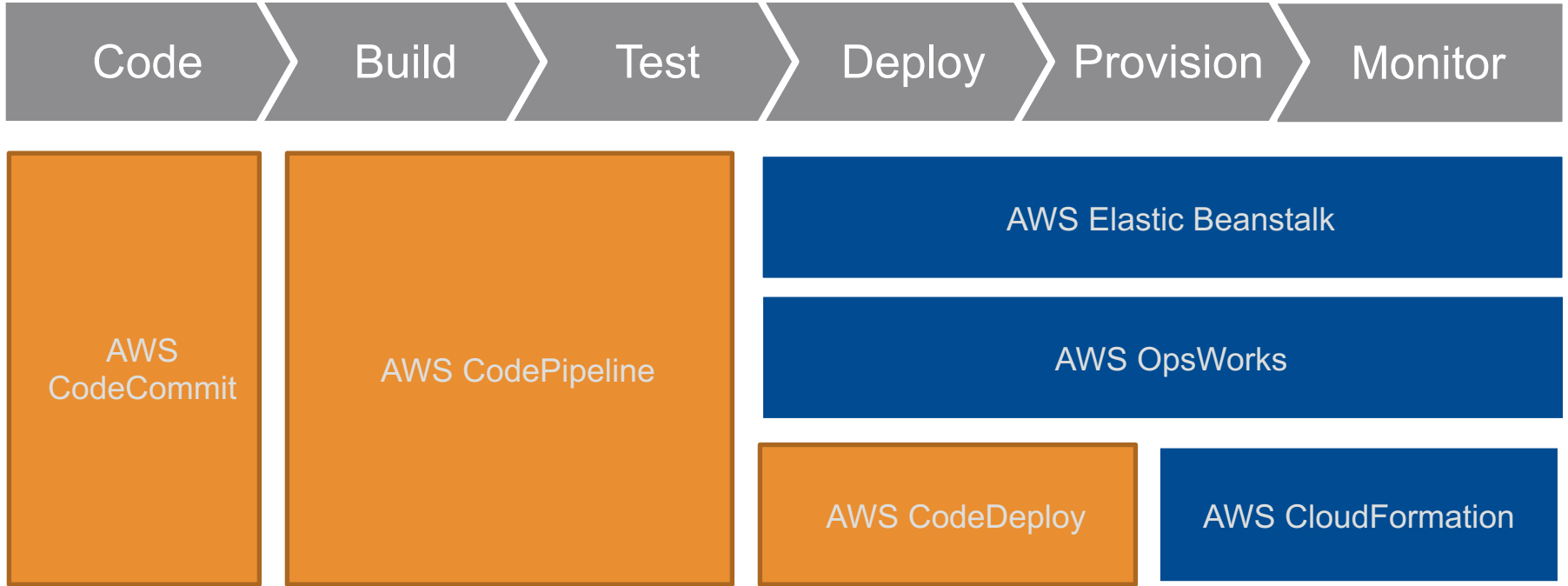
Multi-tenant Deployment



SaaS Deployment Tenets

- Agility, agility, agility
- Minimize tenant variation
- Centralize, version, and control tenant configuration and deployment
- Share and extend deployment automation wherever possible
- Prefer more granular deployments
- Never accept downtime
- Invest in agile data migration and versioning strategies
- Expect to build and continually evolve your own tooling
- Repeatability = stability = agility

AWS DevOps Tooling



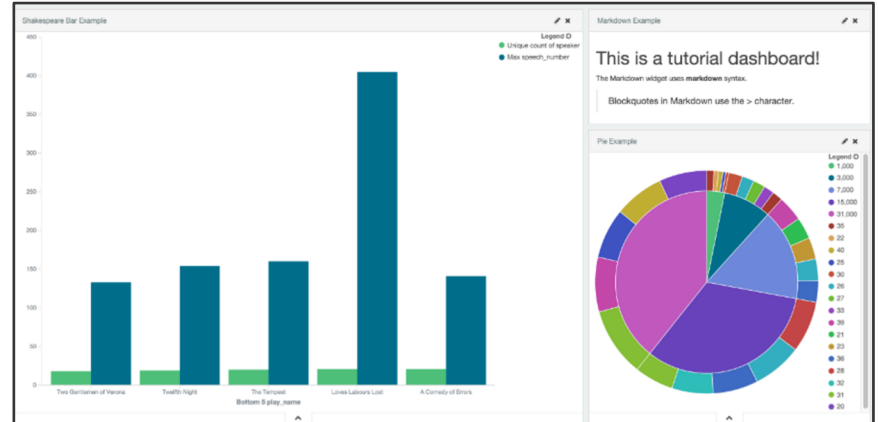
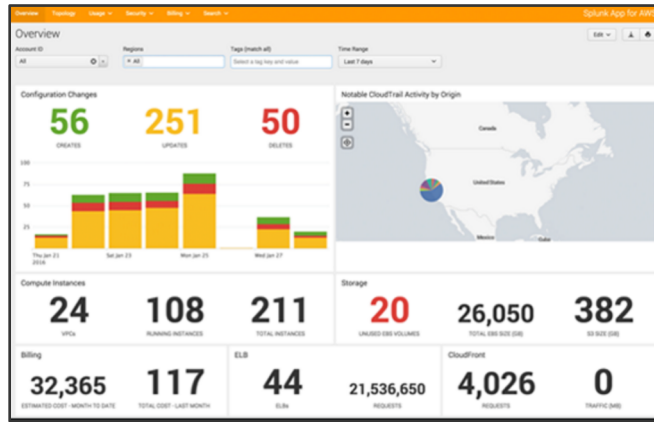
Testing SaaS Solutions

- Profiling & modeling tenant loads
 - Building an application scaling profile and SLAs
 - Simulating tenant loads, elasticity and application failures
 - Validating pricing/metering tiers
- Security & isolation testing
 - Validating role-based access limits on cross-tenant access
 - Validating tenant isolation schemes
 - Assessing data security constraints
- Validating storage scaling
 - Testing IOPS throughput and policies
 - Testing bulk operations

Manage

Best practices, tools, and techniques that are commonly applied to manage a SaaS environment

Monitoring & Management



- SaaS sets a higher bar for management & monitoring
- Views of both infrastructure and application health
- Ability to view health and activity with tenant context
- Instrumenting logs with application and tenant metrics
- Support for tenant policies for alerts/alarms

SaaS Support Considerations

- SaaS demands a robust support solution
- Provider tools that streamline the support lifecycle
- Instrumenting applications with support hooks
- Mapping support SLAs to tiered pricing plans
- Creating a cohesive customer experience that connects internal and AWS support processes

System Administration

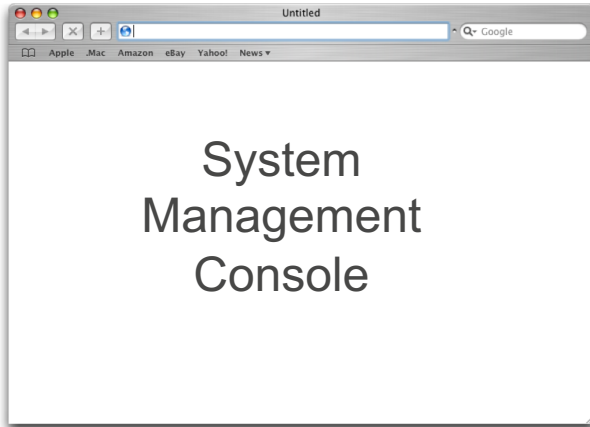


System Admin

Authentication &
Authorization



System Admin User
Management (RBAC)



- Tenant lifecycle management
- Tenant status/history
- Tenant policy management
- Tenant notification/messaging
- System alerts/alarms

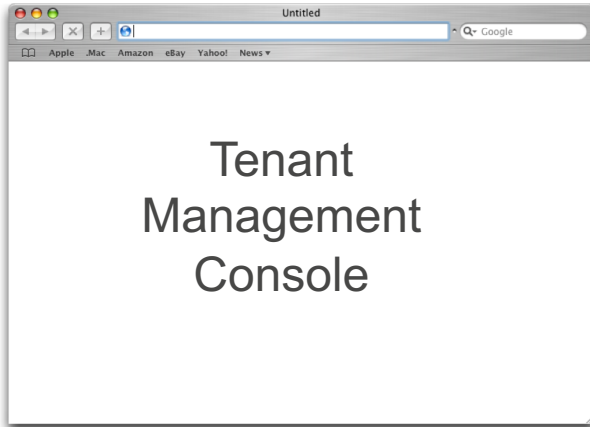
Tenant Administration



Tenant Admin

Authentication &
Authorization

Tenant Admin User
Management (RBAC)



Tenant
Management
Console

- App user management
- App user activity tracking
- App policies and configuration

Profile

Continually evaluating and characterizing tenant consumption and activity.

Profiling Sources

- Usage analytics
 - Insights into application usage data and patterns
 - Ability to assess global and tenant-centric activity
 - Identify paths and hot spots that may be undermining adoption
 - Inform prioritization of features
- Resource metering
 - Correlating tenant activity with resource consumption
 - Identifying resource bottlenecks
 - Profile costs and informing price modeling
 - Capping resource consumption

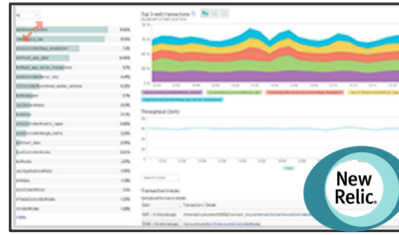
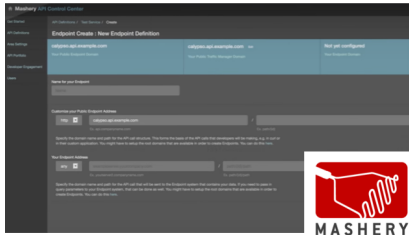
Capturing & analyzing analytics

Page Load Metrics

User Clicks

REST Calls

Application Activity



- Analyze tenant flows
- Assess consumption metrics



App Service Metrics



Storage Metrics



Compute Metrics

Resource Consumption

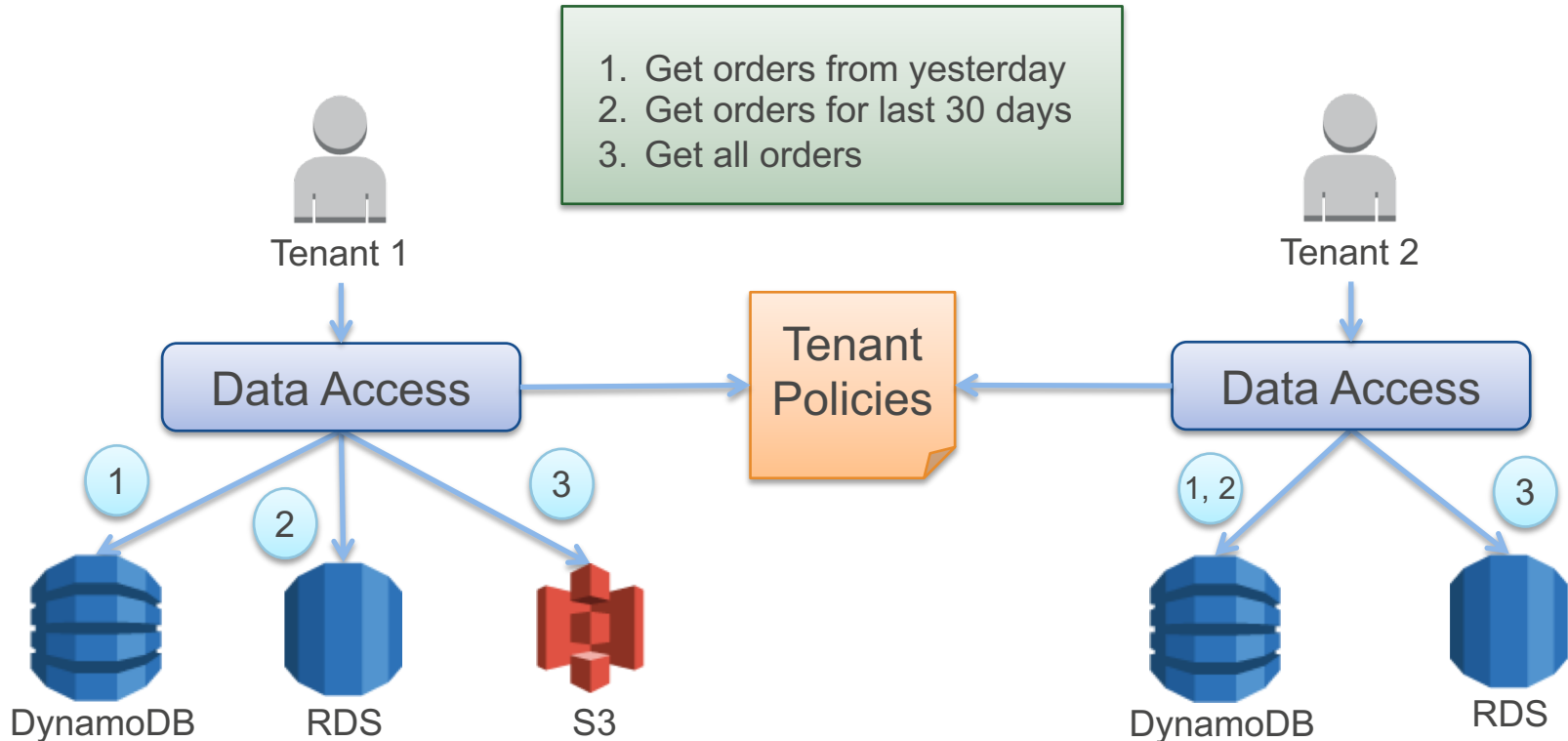
Optimize

Refining the cost and performance footprint of your SaaS application and optimizing target tenants and workflows

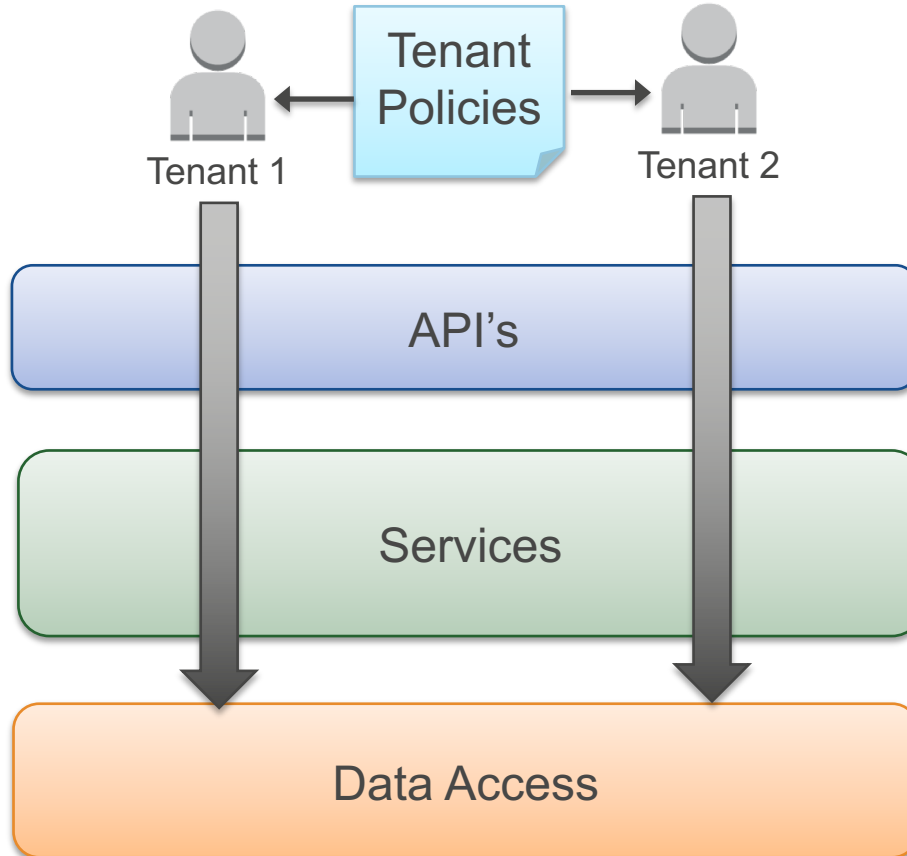
Optimizing SaaS Environments

- Optimizing costs
 - Profiling tenant activity to establish baseline minimum consumption
 - Applying RIs and spot instances
 - Tier optimized service consumption
- Optimizing tenant experience
 - Providing tenant experience optimizations
 - Optimizing based on load/activity
 - Optimizing based on tenant tier
- Optimizing availability and performance
 - Profiling and responding to spikey loads
 - Tuning metrics and scaling policies
- Timing and optimizing bulk operations

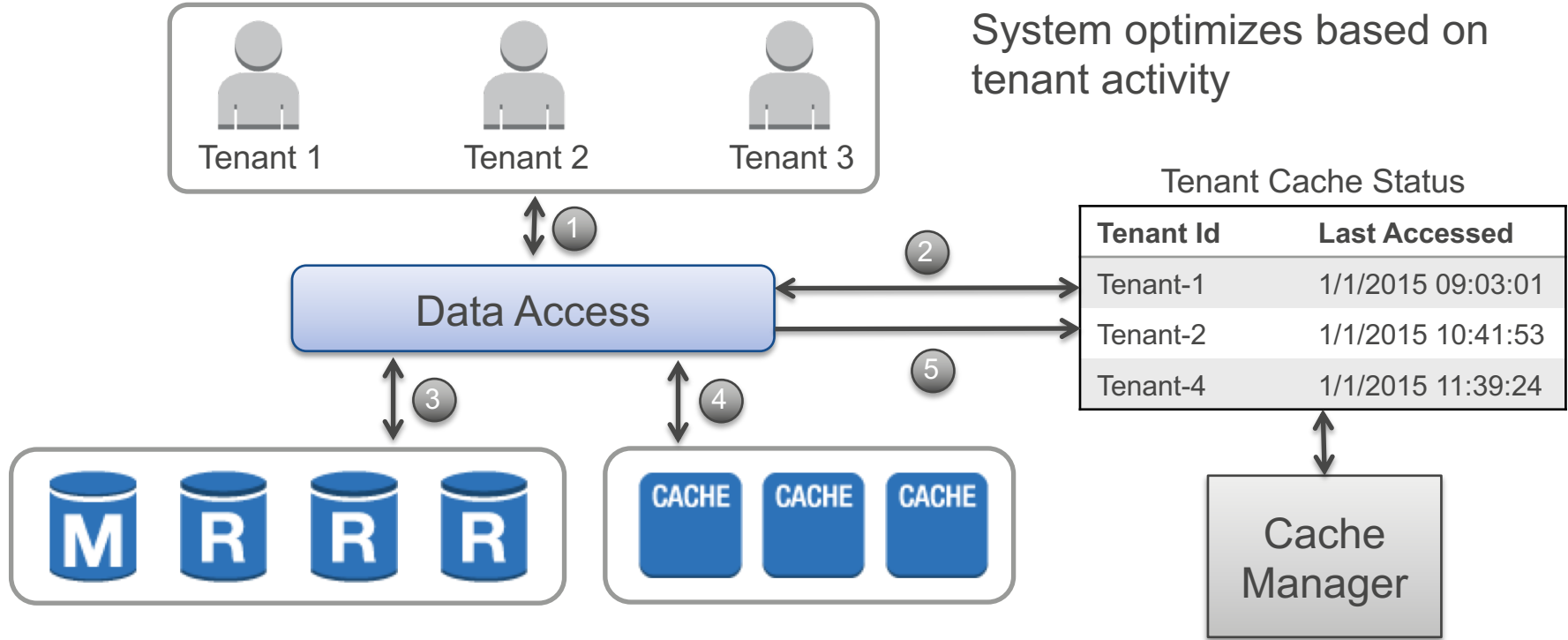
Scenario Driven Storage Optimization



Scenario Driven Storage Optimization



Tenant Load Driven Optimization



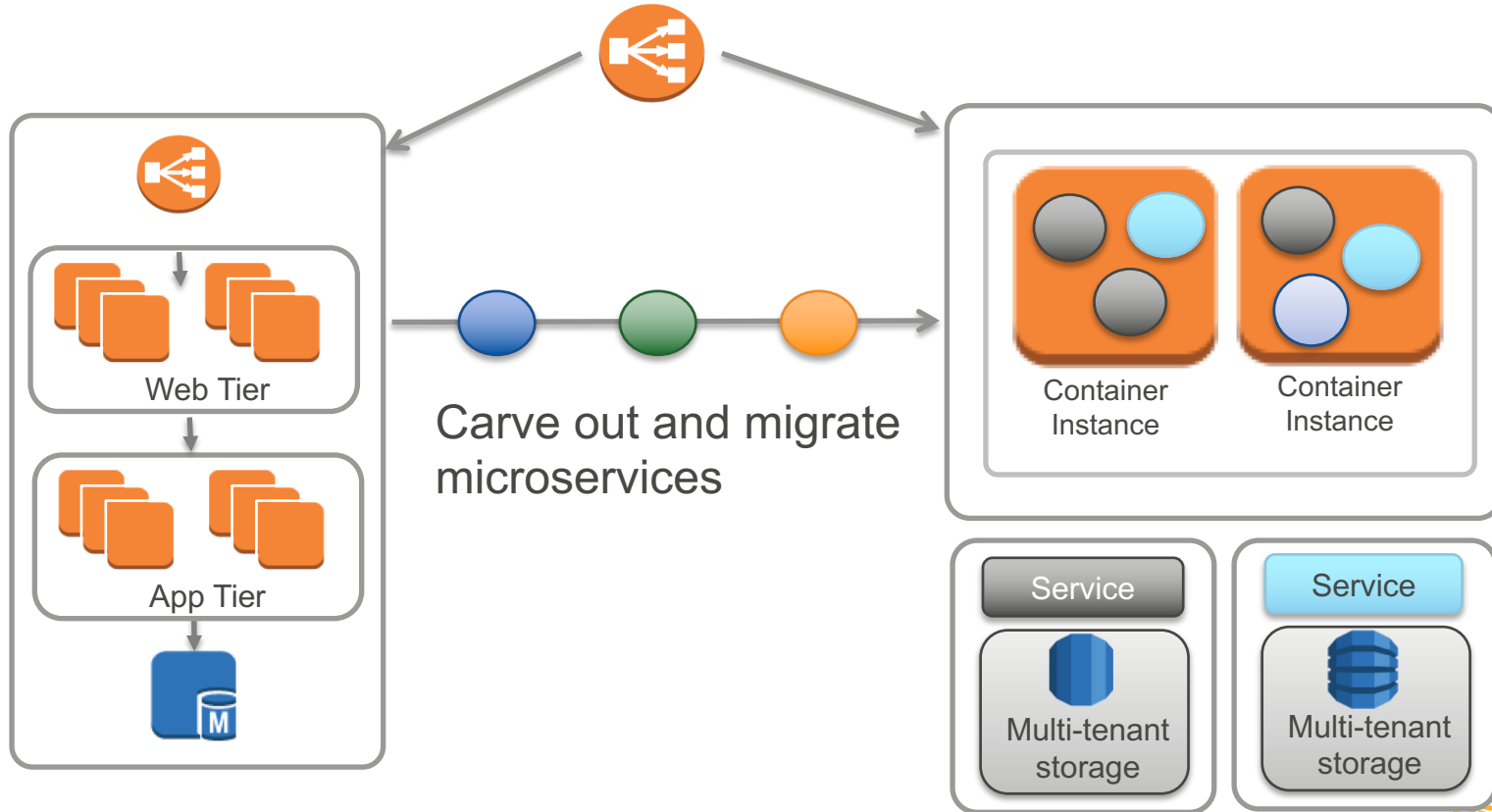
Migrate/Transform

A review of common strategies and value systems that are applied when migrating applications to a SaaS model

Migration Patterns

- Silo tenant migration
 - Existing app moves as-is and is deployed as isolated single tenants
 - Billing, metrics, and analytics surround existing stack
 - Tenant context instrumented into logging, metrics, etc.
- Layered migration
 - Layers of the application stack are iteratively moved to multi-tenancy
 - Gradual introduction of new, multi-tenant app services
 - Storage migrated to multi-tenant model
- Data migration
 - Migrating data to multi-tenant models

Microservice Hybrid Migration



Operational Migration Considerations

- Adopting new on-boarding models
 - Transitioning to multi-tenant authentication/authorization
 - Integrating billing support
- Migrating to tenant aware operations
 - Adding tenant context to management & monitoring
 - Introducing tenant aware logging
 - Adding support for tenant configuration & management
 - Retrofitting metering and analytics
- Business migration
 - Capturing tenant profiles and defining tiered pricing models
 - Customer transition strategies

Sell

Patterns for packaging and promoting SaaS offerings with emphasis on accelerating customer acquisition

Market

- Innovation sandbox
 - Credits
- Test drive
- Go-to-market resources
- Lead generation campaigns
- Marketplace presence

Guiding Principles

A high level collection of SaaS focused principles that should guide and shape any SaaS solution

People, Process, and Culture

- SaaS is a lifestyle
 - Adopting a fail fast approach
 - Customer feedback and product strategy happens in real-time
 - Take pride in your ability to pivot and react
 - Expect your business model to be fluid
 - Drive loyalty through rapid response and constant evolution
- Technology and process must be enablers
 - Agility must be baked into architecture, design, and deployment
 - Migration must be constant and painless
 - Zero tolerance for any notion of down time
 - Ability to package offerings that align with emerging tenant profiles

Common Pitfalls

- Making analytics and metering and afterthought
- Under-investing in management & monitoring
- Allowing noisy tenants to undermine overall experience
- Coarse-grained services undermine scale and cost footprint
- Not separating data read from data write
- Deprioritizing SLA policies
- Deployments that require down time
- Underestimating the cultural dimensions of SaaS
- Under-investing in deployment automation and testing

