A MORE FLEXIBLE MULTI-TENANT SOA FOR SAAS

Eric H. Nielsen, Ph.D.
VP Platform Architecture
CA Technologies
e.h.nielsen@ieee.org

For IEEE Software Technology Conference
STC 2014 April 3, 2014 Long Beach, CA USA

Abstract

Like many vendors, CA has multiple existing products with stand alone value. What does the road to SaaS look like for these apps? How do we make them work together as a suite? Applications may be sourced from multiple suppliers, deployed from multiple PaaS, or incorporate different generations of architecture. Before enabling functional integration among heterogeneous applications, we need to determine how tenancy will work.

A typical multi-tenant SaaS application scales horizontally as a single composite app. with controlled integration APIs. The SaaS application is single-sourced, and tenancy is wired into the application. Most enterprise apps aren't built to be deployed this way; it is more than building for scale and integration APIs, it is a matter of how each application adapts to deal with tenancy.

Unlike the prototypical SaaS application, this problem demands loose coupling between the logical service layer and the physical resource layer. And to be business rational, traditional applications have the additional challenge of merging current software and business practices incrementally into SaaS technology.

To do this we had to rethink tenancy in terms of customers and environments, and how these environments are represented at the service request layer and kept external to each application. Environments provide walls for access, content, and SLAs. This presentation summarizes resulting architecture is based on our experience creating a multi-tenant environment service layer to deliver IT management applications as a SaaS suite, as well as proposes specific areas where standards are recommended to ensure open implementation.

…built for MultiTenancy?

Source: http://upload.wikimedia.org/wikipedia/commons/thumb/8/84/Joshua_P_Young_House.JPG/640px-Joshua_P_Young_House.JPG

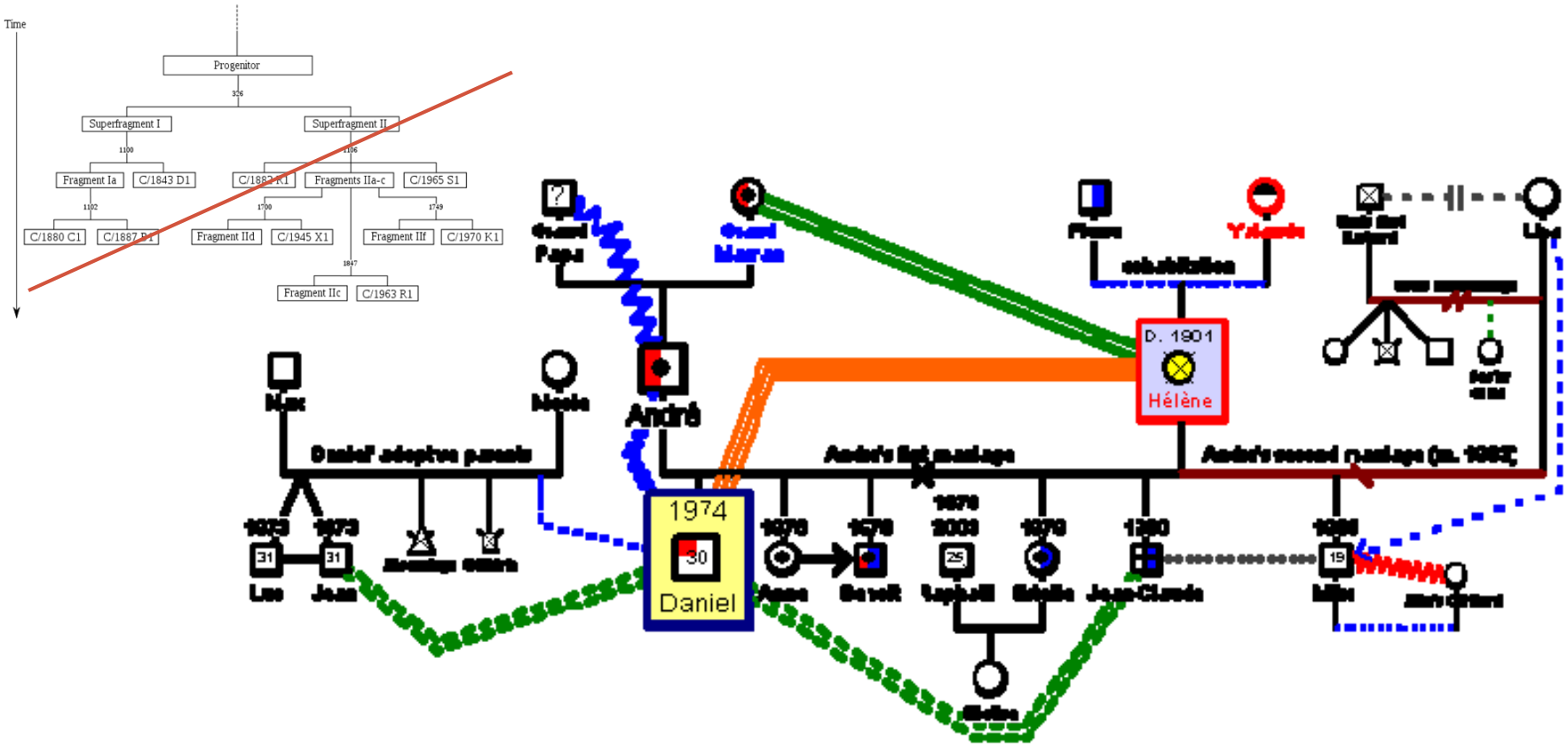
Thrive! Embrace multi-tenancy!

Source: http://en.wikipedia.org/wiki/File:View_of_the_Empire_State_Building_from_the_Rockefeller_Center_observation_deck_NYC_-_18_August_2009.jpg

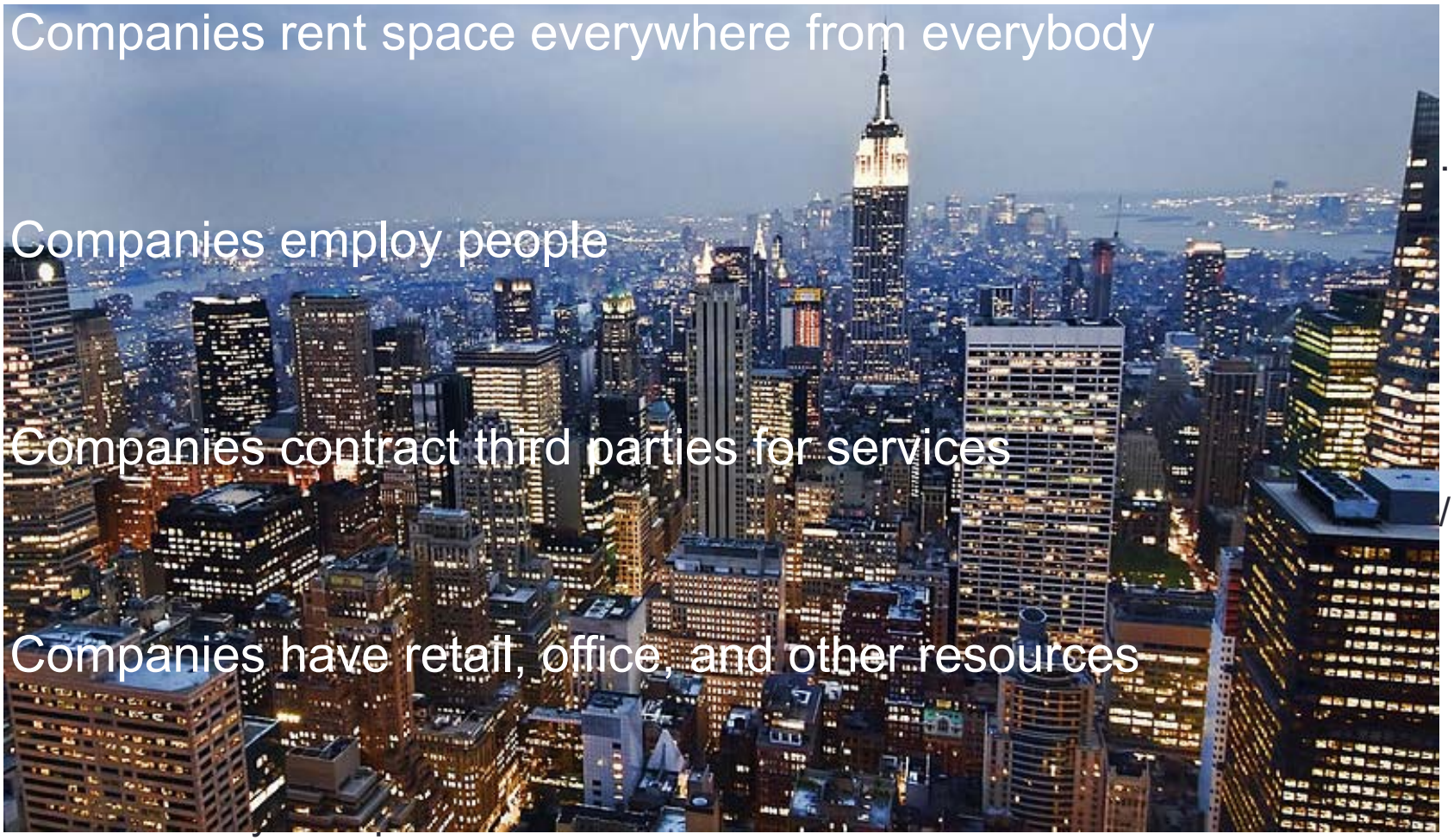
My Background

- building multi-tenant SaaS Apps for 14 years.
- building multi-tenant SaaS that spans suites of single and multi-tenant Apps for 4 years.
- defined software tenancy to meet business needs.

Tenancy is ~~hierarchical~~ complicated



Tenancy needs to be simple



Tenancy Definitions

Google AppEngine: "*Multitenancy* is the name given to a software architecture in which **one instance of an application**, running on a remote server, **serves many client organizations** (also known as *tenants*)."

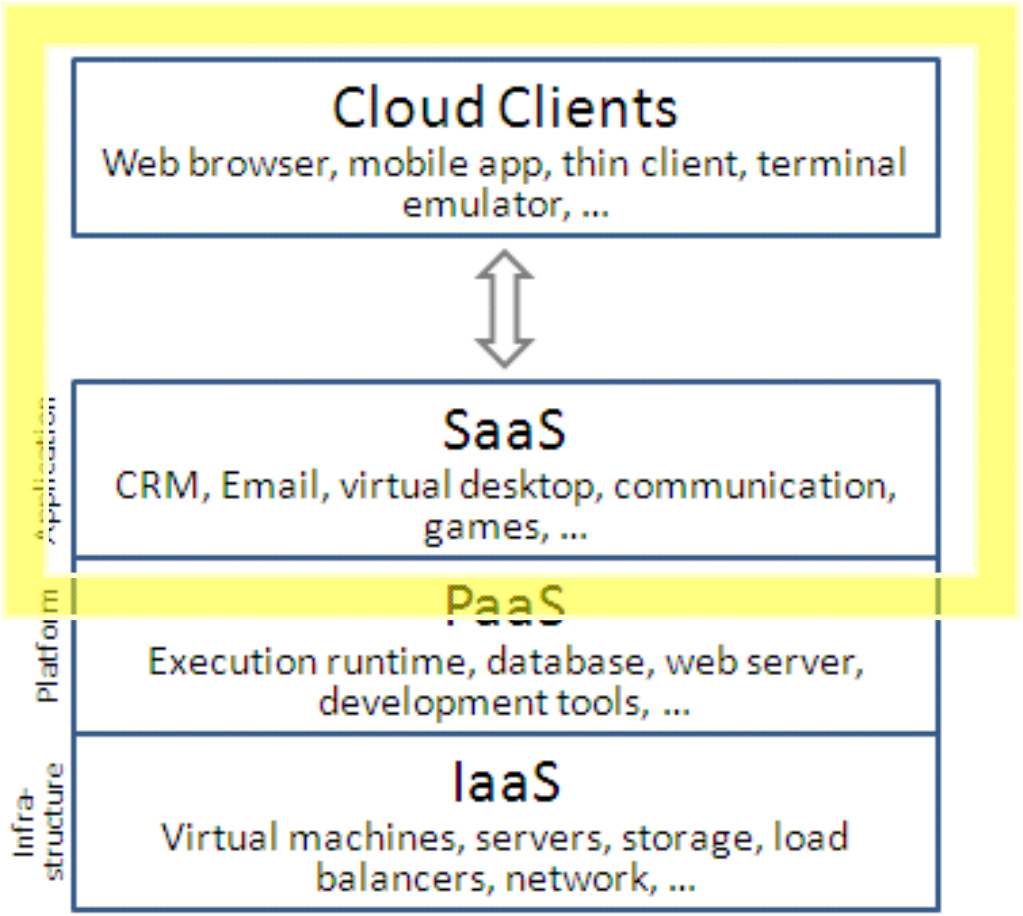
WSO2 (Chris Haddad): "**Tenancy dimensions**" …. "consider whether the SaaS **application will be personalized** across people (e.g. organization, team, or individual), projects (e.g. joint venture, marketing campaign, benefits enrollment), or IT artifacts (e.g. application, API, service, or machine)."

Wikipedia: "Multitenancy refers to a principle in software architecture where a **single instance of the software** runs on a server, serving **multiple client organizations** (tenants). "

Expand tenancy to include the customer perspective!

Tenancy at each level should be decoupled from other levels

This presentation is exclusively about the tenancy at the service layer



From: http://en.wikipedia.org/wiki/File:Cloud_computing_layers.png

Tenancy Use Cases that are hard

- Converting a Customer from a trial to production
- Recreating integrated sandbox environments.
- Customer reorganization and changing purchasing silos.
- Data ownership and tracking external to applications.
- Simplicity of integration, workflows and custom solutions within an environment.
- Users using or administering multiple environments.
- Operationally tracking data and enforcing legal limitations.

Flexible MultiTenant SOA:

Technical Behaviors

- Discoverable APIs and automatic integration.
- Environment Replication without conflict.
- Custom configuration of a single solution..
- Resources referenced independent of source and Auth.

Solution:

➔ **model** tenancy in terms of customer and environment

➔Use API **URIs** that make environments explicit

Migration

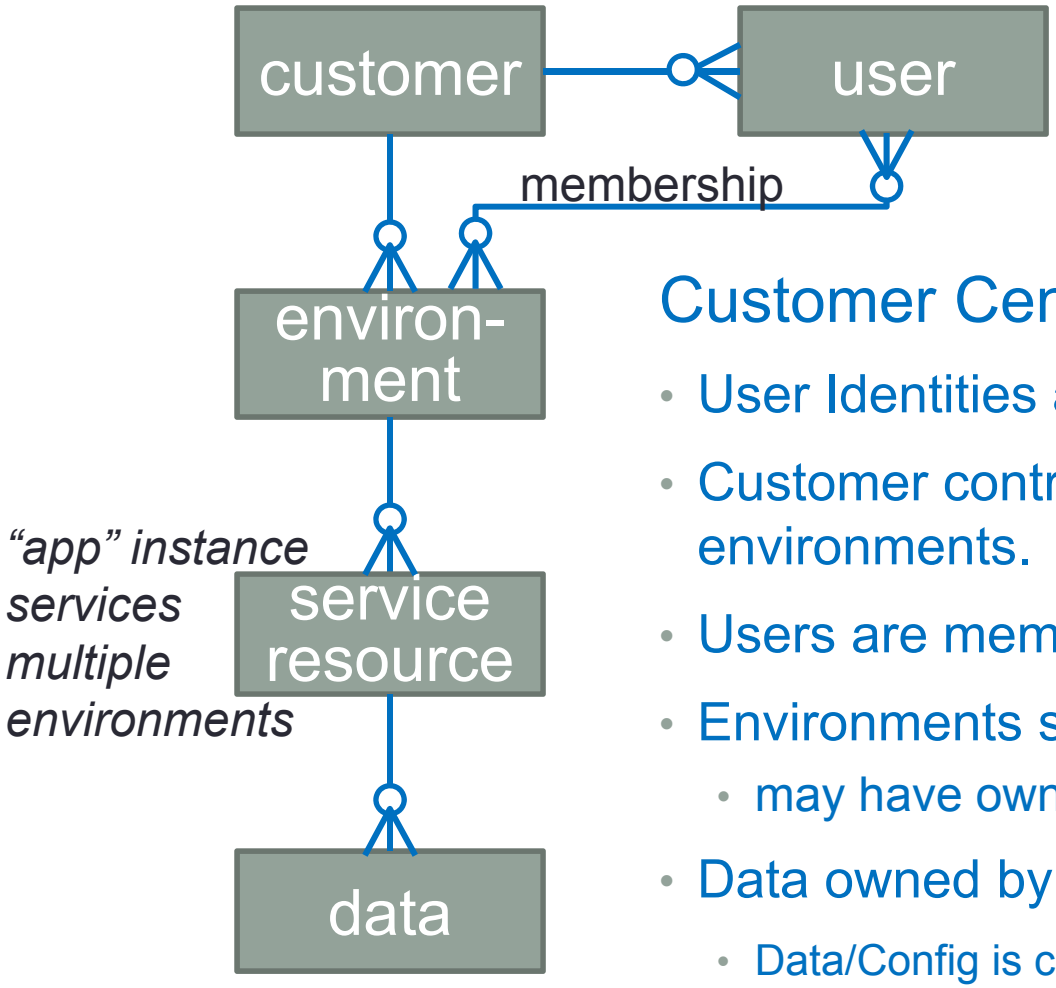
➔ implement atop existing APIs.

Tenant = 1 Customer with ~~1~~ *N* Environments

Customer Environments at the Service Layer

- move copies of data between environments,
  do not mix results among environments.
- have different service requirements by environment
  backup, audit, recovery, costs

Tenant Refactored:
One Customer, Multiple Environments

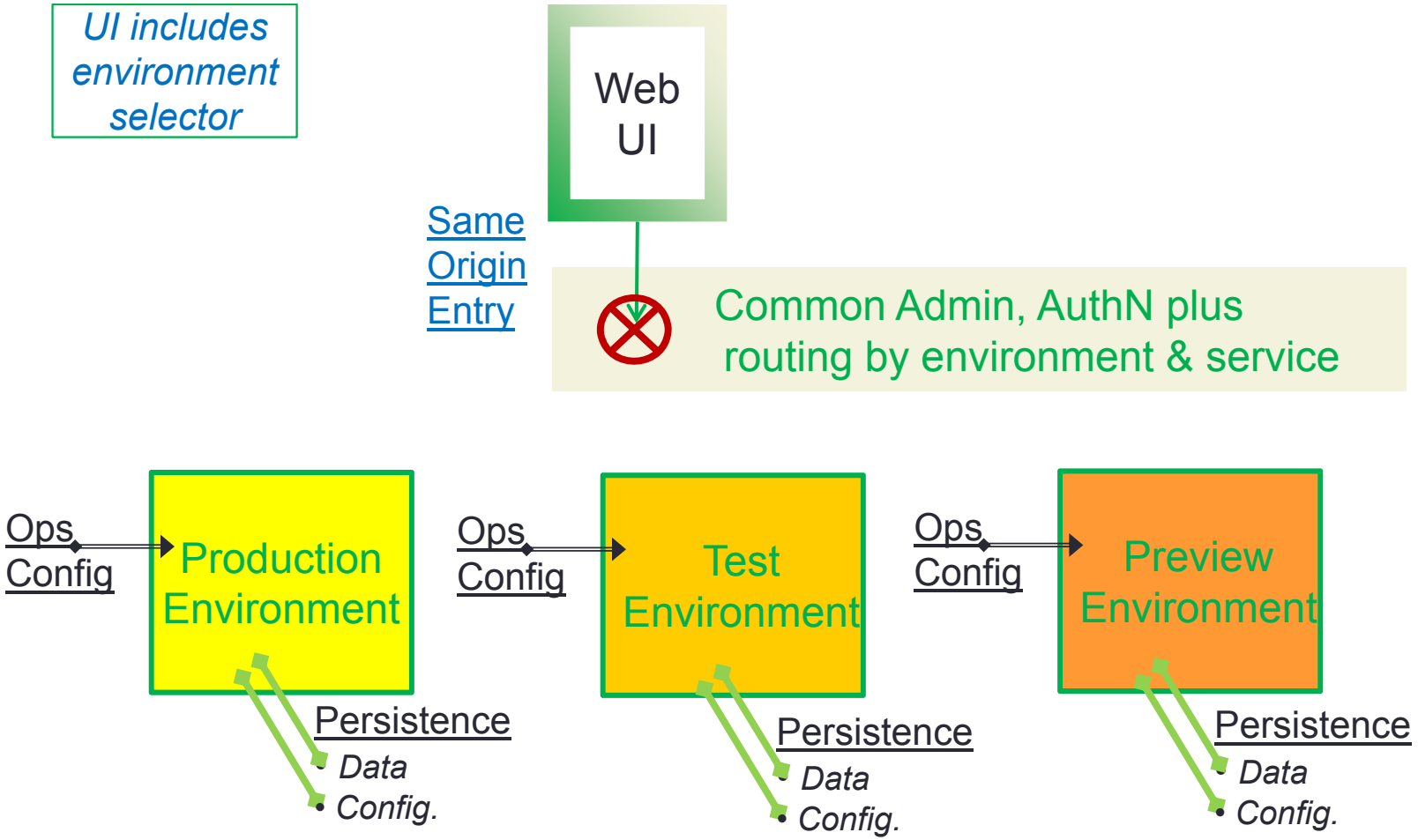


Customer Centric Model

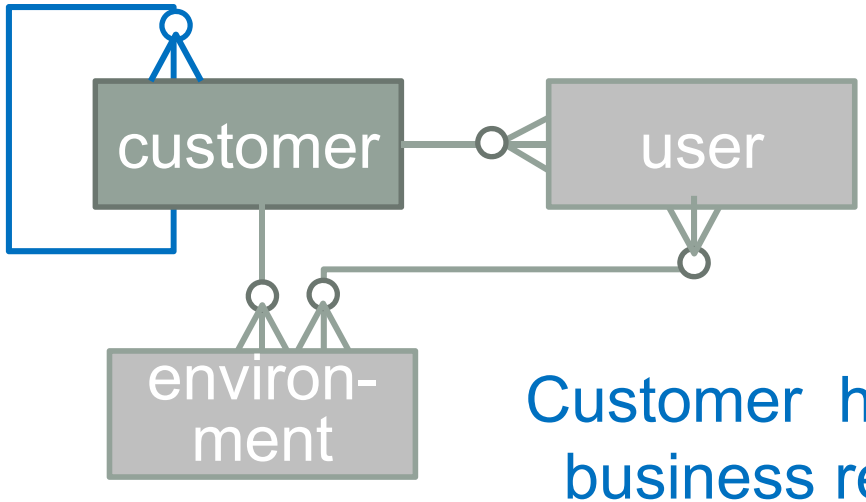
- User Identities are owned by the customer.
- Customer contracts services for zero or more environments.
- Users are members of multiple environments
- Environments serve different business purposes.
  - may have own service level, legal restrictions
- Data owned by customer, walled by environment
  - Data/Config is copyable between environments.

Environment Oriented Tenancy

➔ Multiple Environments per Customer



Tenants Refactored:
Customers are Hierarchical and Flexible



Customer hierarchy represents business relationships

- Hierarchy represents who is visible to whom, who can administer whom, licensing, etc.
- Customers can be split, merged, and moved with no functional effect on environments

Environments map to Operational Characteristics

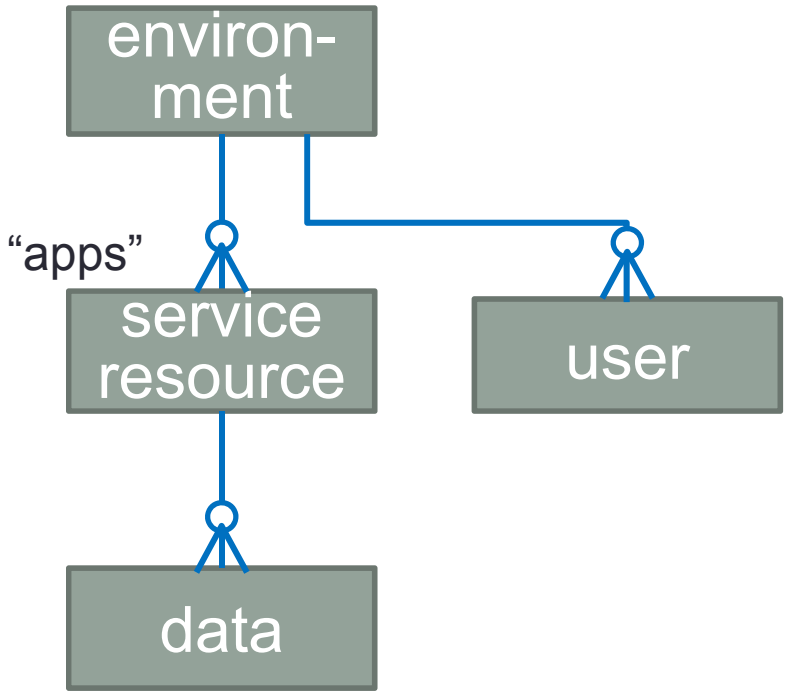
Environ-ment	SLA	Maint. Window	Data Backup	Data Resiliency*	Disaster Recovery	National Boundary
Production	99.95%	no	Daily	point in time	Guaranteed	EU only
Devel.	99%	yes	OnDemand	recoverable	no	EU Only
Test	99%	yes	OnDemand	recoverable	no	EU Only

* Consistent backup across an environment is non-trivial

Tenancy Refactored:

Apps provide services by ~~tenant~~ environment

Apps manage data access…



For app developers "customer environment" is indistinguishable from "tenant"

Apps can discover other resources and services within their environment.

Each Environment provisions app instances to match operational requirements such as availability, backup, data residency.

Longer term: Each environment can create and manage its own, custom solution UI

Environments enable and limit integration

Pixar / Marvel Comics
An Unfortunate Integration!

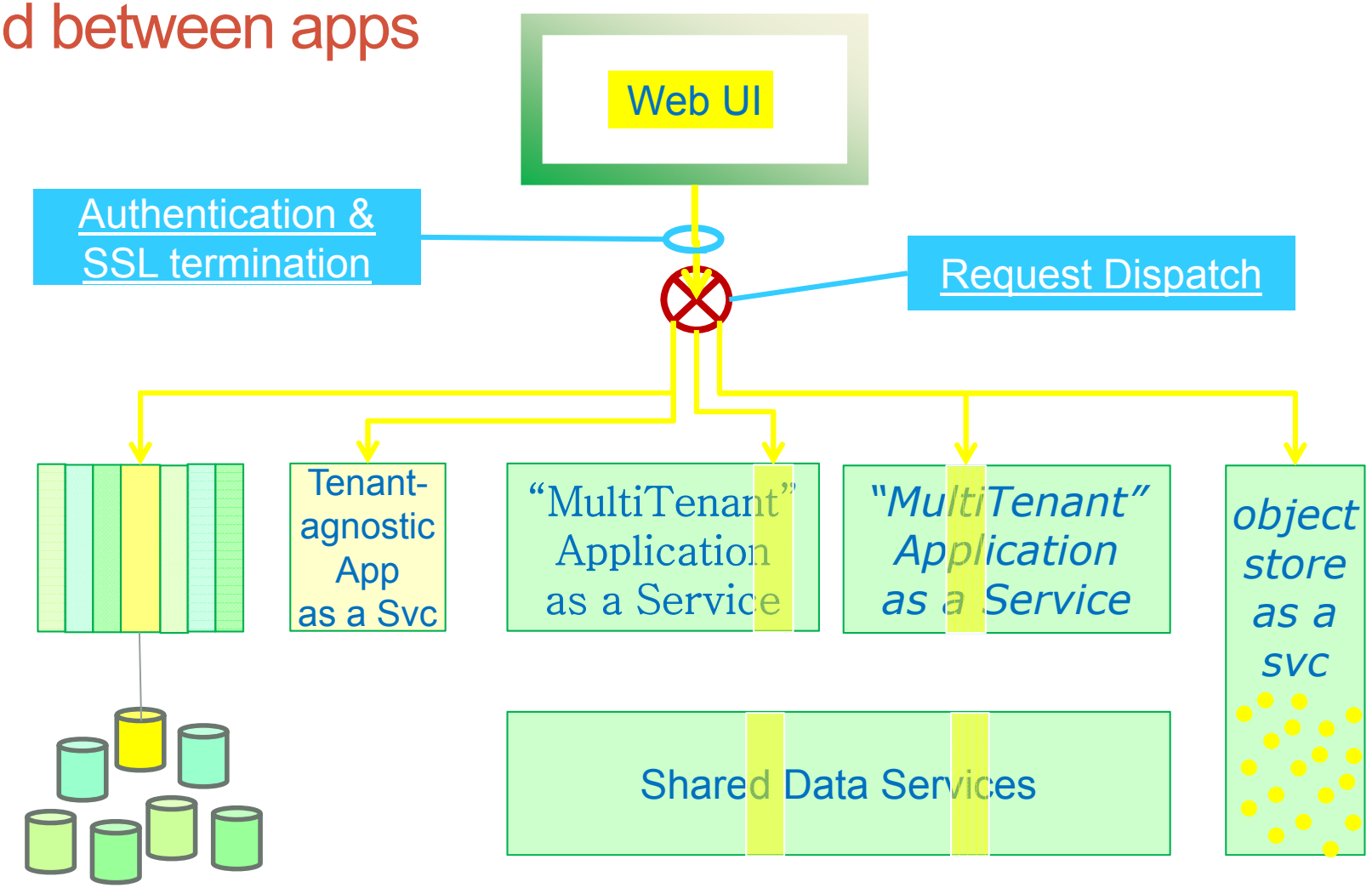
From: worth1000.com

Make integration among intra-environment resources simple.

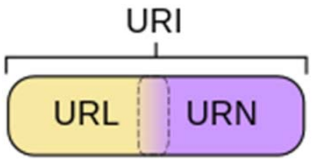
- Discovery and store intra-environment resource references.
- Environment UI navigating among multiple app resources.
- Systematically ensure that Mistakes of Omission do not expose data to another environment
- Advanced: User perform actions in multiple environments simultaneously

Multi-Environment SOA Architecture

requires enhanced request routing from outside and between apps



Standard URIs



From wikipedia:

URIs can be classified as locators (URLs), as names (URNs), or as both. A uniform resource name (URN) functions like a person's name, while a uniform resource locator (URL) resembles that person's street address. In other words: the URN defines an item's identity, while the URL provides a method for finding it.

`Format: <scheme>://<authority><path>?<query>`

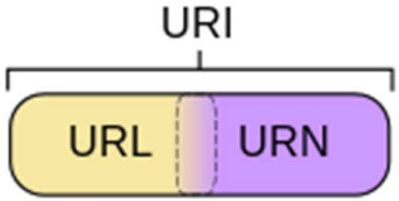
From: RFC 2396

Request includes 3 things…

1. Location of resource;
2. Authentication of user; and
3. Resource being accessed.

+ optional payload (payload resources have environment in URL)

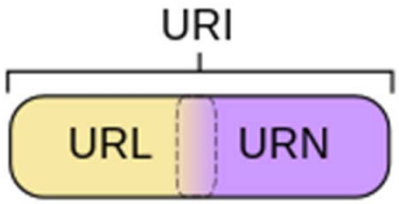
What do we need in a multi-environment URI?



- Resource references are globally unique
  - URI does not include implicit context.
  - Does not contain requester authentication information
- Service Routing for federated services in an environment
  - Each resource type may be implemented by more than one service, such as resource types “user” or “ticket”
  - Different Environments may be services by a different app instance

  ➔Therefore need to route using *environment* & *service type* info.
- Standards and Conventions:
  - RFC 2396 <scheme>://<authority><path>?<query>
  - HATEOAS
  - Avoid URI rewrites
  - Hostname: trusted location of service router

Request Authentication

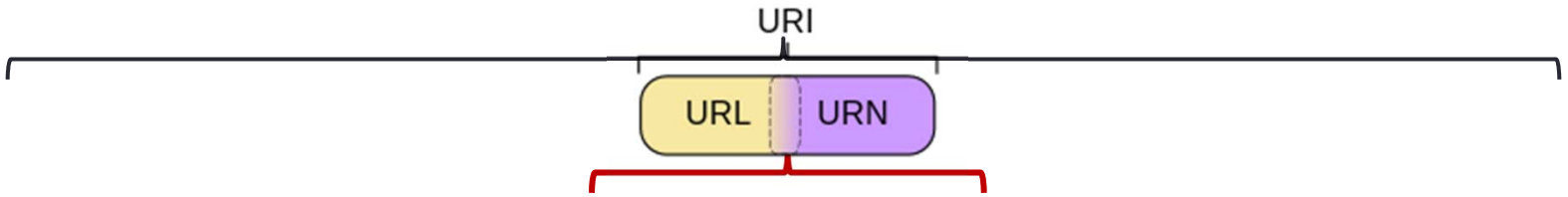


- To determine Authorization, every request needs:
  1) The requesting User; and
  2) The Environment governing request Authentication

➔AuthN distinct from resource

- Extract AuthN User and Environment from request header when proxied from trusted router,
  or
- App validate tokens in each request with a security service.

Implementing a Flexible MultiTenant SOA
One solution for the URI



<scheme>://<authority><service path><resource path>?<query>

RFC2396 <path>

Details:

<scheme>://<authority> ➔ e.g., https://host.com:443

<service path> ➔ **/<proto>/<service id>/<version>/<env. id >**

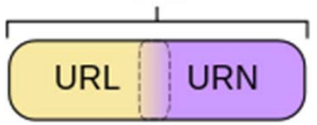
- **<proto>** identifies the routing mechanism
- **<service id>** identifies the service type
- **<version>** identifies the interface version of the service
- **<env.id>** identifies the environment specific resource service

<resource path> ➔ **/<resource type>[/<resource id>[…]]**

Implementing a Flexible MultiTenant SOA

URN uniqueness = no relying on hostname!

<service path> ➔ `/<proto>/<service id>/<version>/<env id >`



Why a <service path>?

- Multiple independent services can implement the same resource type
- Resource references stored/compared independent of <authority>
- Everything in an environment accessible through a single origin.

Now the SaaS infrastructure can…:

- Can replicate data wholesale between environments safely
- Can access data from multiple environments with a single service.
- Can federate single, multi-tenant applications, and tenant-agnostic apps
- Can route requests based on service type and environment id.

Implementing a Flexible MultiTenant SOA AuthN solution for the URI

- All access controls require environment information

Every Request has **`headers`** containing:

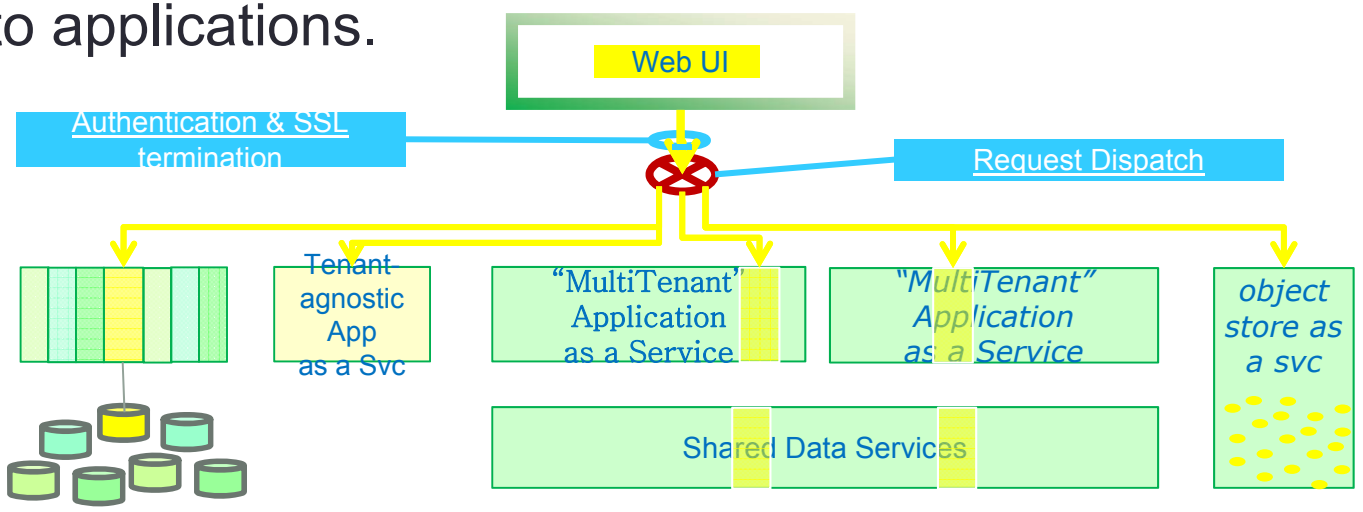
  - AuthN User ID – *apps must use this for AuthZ*
    [may require UserID string mapping]
  - AuthN Environment ID – *apps must use this for AuthZ*
    [salesforce.com example: see the "LoginScopeHeader"]

- When does 'AuthN Env.'$_{\text{in the header}}$ ≠ 'resource Env.'$_{\text{in the URI}}$?
  - When viewing resources from multiple environments

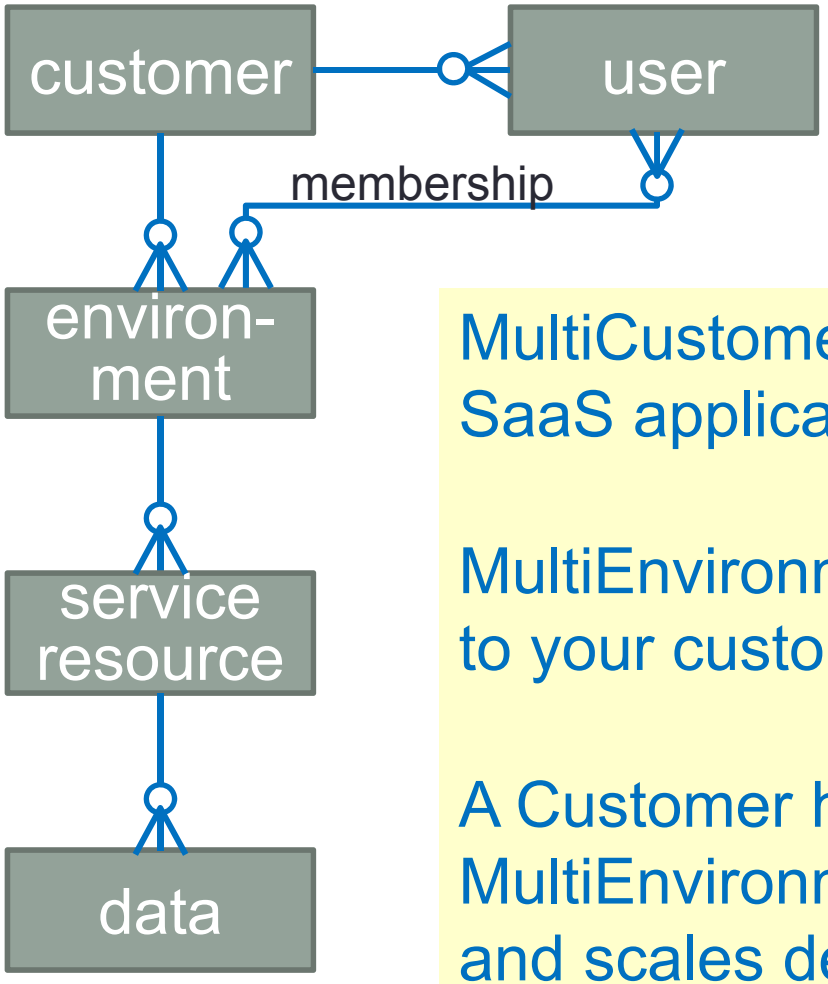
- Legacy: Multi-tenant apps assume user belongs to one environment.
  - Treated as a provisioning limitation.

Summary: Use Cases

- Converting a Customer from a trial to production
- Recreating integrated sandbox environments.
- Customer reorganization and changing purchasing silos.
- Data ownership and tracking external to applications.
- Simplicity of integration, workflows & custom solutions within an environment.
- Users using or administering multiple environments.
- Operationally tracking data and enforcing legal limitations.



Summary: Modeling Tenancy Flexibly



MultiCustomer is critical for any successful SaaS application.

MultiEnvironment management is critical to your customer's experience.

A Customer hierarchy together with MultiEnvironment management simplifies and scales delivery of diverse SaaS applications and deployments.

Summary:
Standardizing the Service Request

Need broad recognition that every request may have:

➔ One Environment ID that governs Authorization of a request

➔ A second (occasionally different) Environment ID that Owns the Resources referred to in a request.

…and to standardize:

➔ Incorporating the owning Tenant-Environment ID in every resource request URI for routing.

➔ Incorporating an Authenticated Tenant-Environment ID header to govern authorization policies applied to a request.