Halil İbrahim Kalkan
Co Founder of Volosoft

GitHub: **@hikalkan**
Twitter: **@hibrahimkalkan**
Email: **halil.kalkan@volosoft.com**

Multi-Tenancy

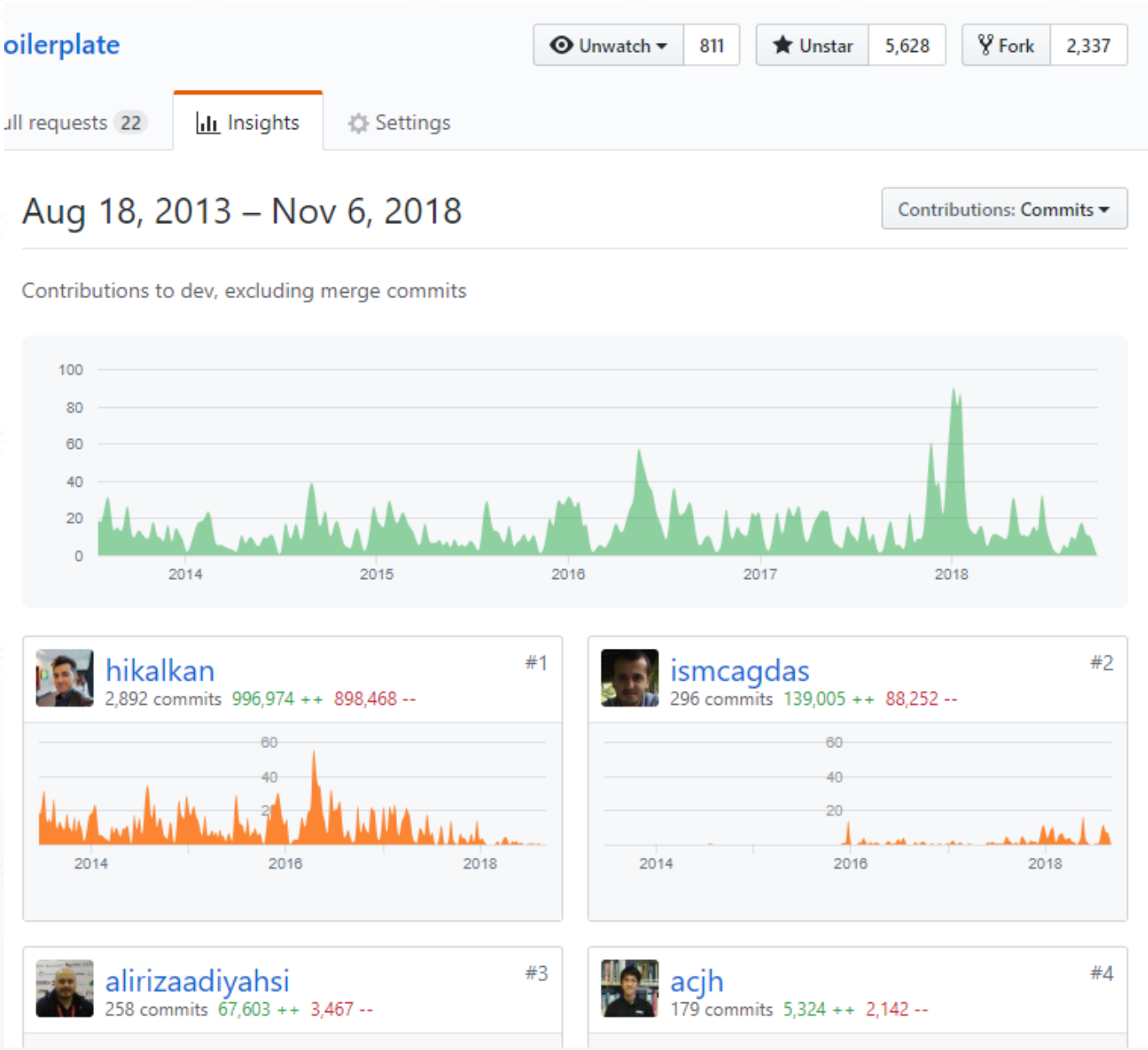
Development, Challenges & Solutions.

ASP.NET Boilerplate

aspnetboilerplate.com

- Multi-Tenancy
- Modular & Layered Architecture
- Domain Driven Design
- Free & Open Source

- 5+ years of continuous development.
- 5,500+ stars on GitHub.
- 700,000+ downloads on NuGet.



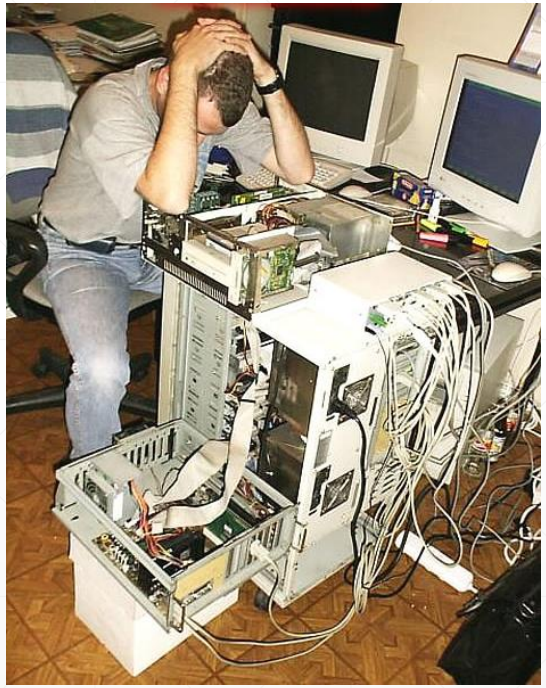
Agenda

- SaaS, multi-tenancy, benefits & difficulties.
- Multi-Tenancy: High level architecture / database separation.
- How to determine & change the current tenant.
- Implementing database & data isolation.
- Temporarily enable/disable multi-tenancy.
- Handling database migrations.
- SaaS features and declarative feature check.
- Summary

SaaS

Software as a Service

On-Premise / IaaS / PaaS / SaaS



On-Premises	Infrastructure as a Service	Platform as a Service	Software as a Service
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking
You Manage		Other Manages	

SaaS
Concepts

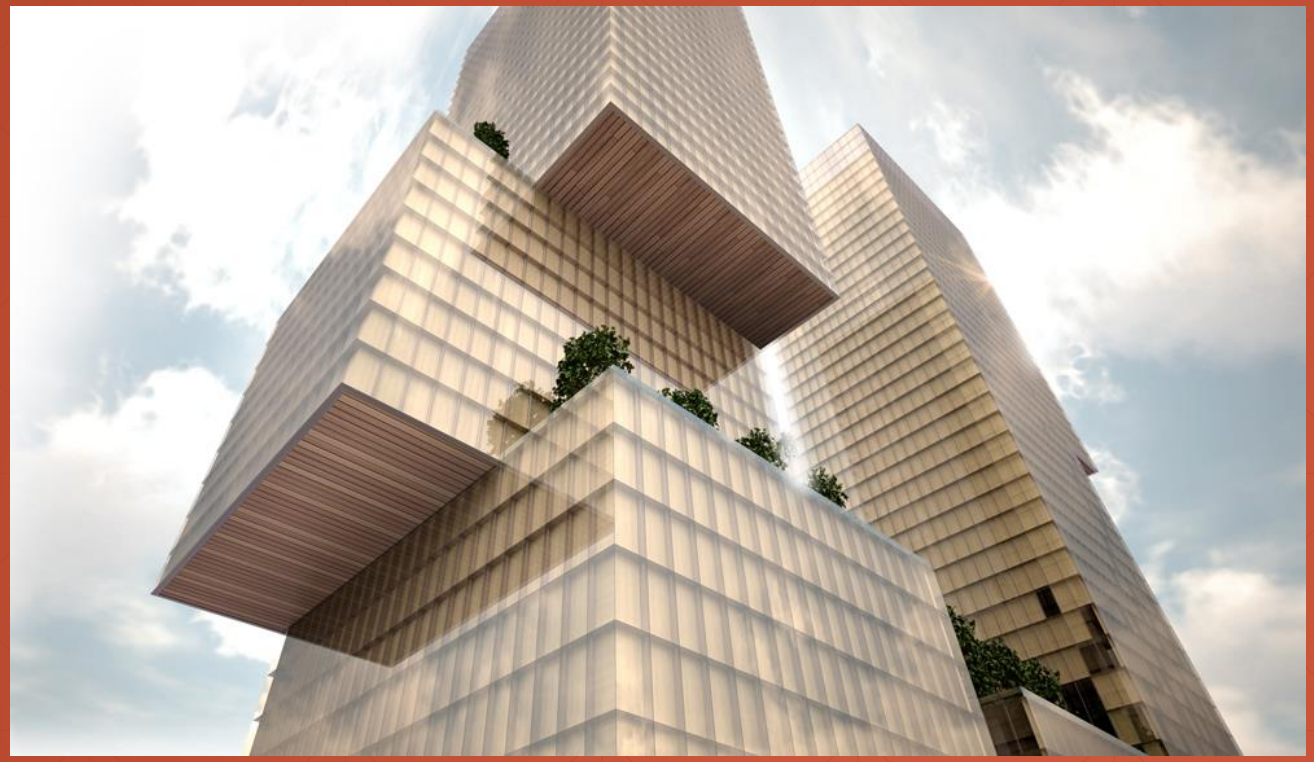
- **Tenant**: An organization that uses the application/service (and pay for it)
- **Host**: The organization that is responsible to provide the service and manage all the tenants

Why Multi-Tenancy?

- Maximum Utilization / Low Costs
- Easy to add a new client (tenant)
- All clients use the same application & version
  - Easier maintenance & upgrade

Difficulties

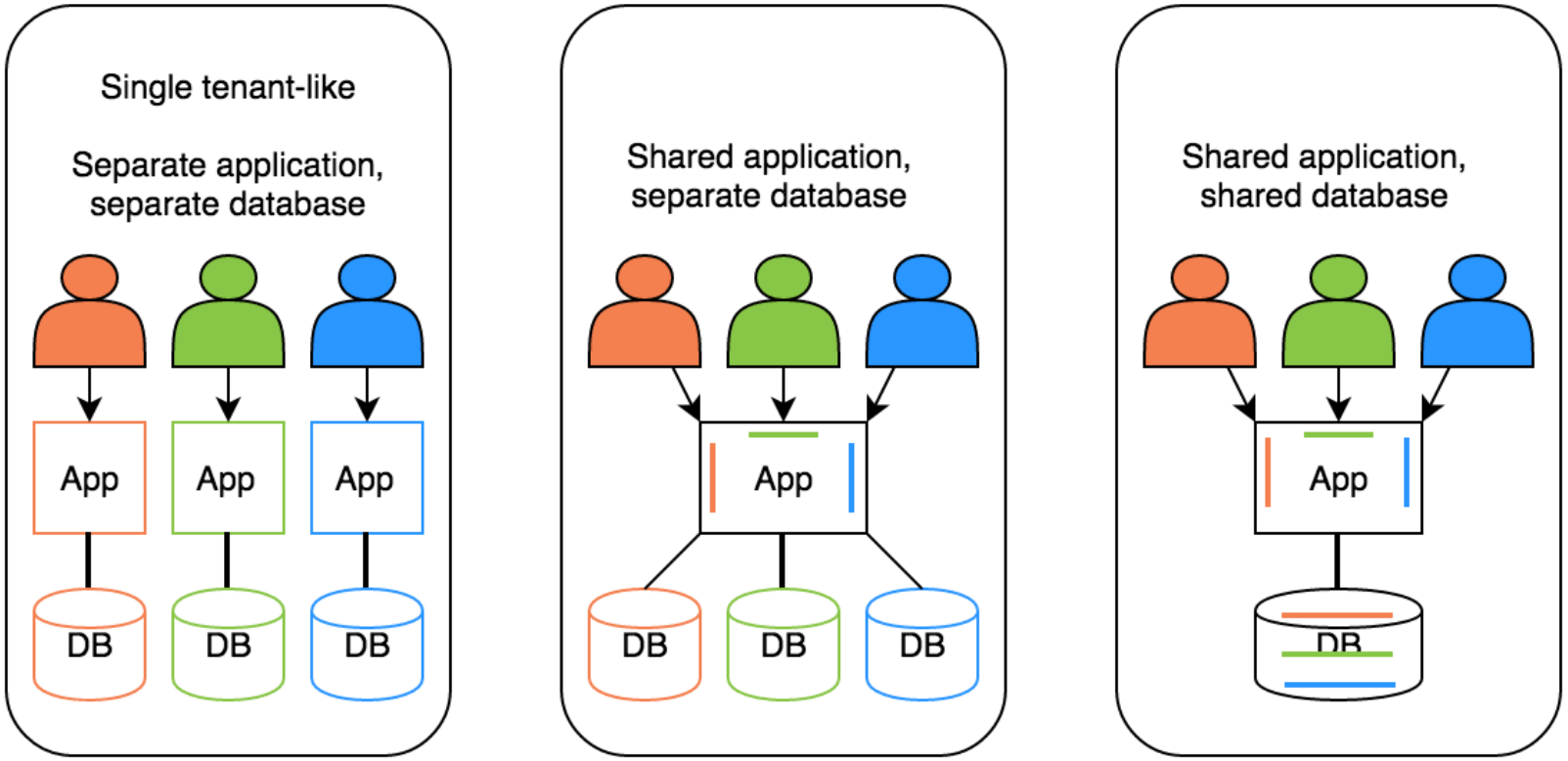
- Data Isolation
- Performance: One tenant’s heavy usage may effect the other tenants
- Configuration & customization per tenant
- Security
- Backup (per tenant)



Multi-Tenancy

High-Level Architecture

Deployment / Database Options



Hybrid!

Ideal Multi-Tenant Application

- Works exactly like on-premise: **Separated users, permissions, data…**
  - Should be able to work as **on-premise** too
- Should be developed **independent from multi-tenancy**

Stateless Application Design

- Application should be **stateless**!
- Main state origins:
  - **Http Request**: Cookie, header, querystring, payload
    - Authentication Ticket
  - **Database**: Relational, non-relational
  - **Cache**: Redis, Memcached

Multi-Tenancy

Determine the Current Tenant

Determine the Current Tenant

- Sources
  - Current **Claims** (if user has been authenticated)
  - **Subdomain** (or domain): https://***tenancy-name***.mydomain.com
  - Http **Header**: _tenant = “acme” (for API clients & SPAs)
  - Http **Cookie**: _tenant = “acme” (for MVC applications)

How to Get/Pass a TenantId?

```
public class DemoWithoutAmbientContext
{
    public List<Product> SearchProducts(string productName, Guid? tenantId)
    {
        return GetAllProducts(tenantId)
            .Where(p => p.Name.Contains(productName))
            .ToList();
    }

    private List<Product> GetAllProducts(Guid? tenantId)
    {
        return GetAllProductsFromRepository(tenantId);
    }

    private List<Product> GetAllProductsFromRepository(Guid? tenantId)
    {
        return GetAllProductsFromDbContext(tenantId);
    }

    private List<Product> GetAllProductsFromDbContext(Guid? tenantId)
    {
        //tenantid is only needed here!
        throw new NotImplementedException();
    }
}
```

Ambient Context Pattern

```
public class DemoWithAmbientContext
{
    public List<Product> SearchProducts(string productName)
    {
        return GetAllProducts()
            .Where(p => p.Name.Contains(productName))
            .ToList();
    }

    private List<Product> GetAllProducts()
    {
        return GetAllProductsFromRepository();
    }

    private List<Product> GetAllProductsFromRepository()
    {
        return GetAllProductsFromDbContext();
    }

    private List<Product> GetAllProductsFromDbContext()
    {
        var tenantId = TenantInfo.Current.Id;
        throw new NotImplementedException();
    }
}
```

Implementation using AsyncLocal<T>

```
public class TenantInfo
{
    public static TenantInfo Current
    {
        get => _current.Value;
        set => _current.Value = value;
    }
    private static readonly AsyncLocal<TenantInfo> _current = new AsyncLocal<TenantInfo>();

    public Guid Id { get; set; }

    public string Name { get; set; }
}
```

Change Current Tenant

```
public class TenantChangeDemo
{
    public Tenant CreateNewTenant(string name)
    {
        var tenant = new Tenant(name);

        using (TenantInfo.Change(new TenantInfo(tenant.Id, tenant.Name)))
        {
            CreateAdminUser();
        }

        return tenant;
    }

    public void CreateAdminUser()
    {
        var admin = new User("admin", TenantInfo.Current?.Id);
        //SaveUser(admin)...
    }
}
```

Change Current Tenant (Nested)

```
public class TenantChangeDemo2
{
    public void DoIt()
    {
        using (TenantInfo.Change(new TenantInfo(Guid.Parse("a745a68e-d316-4828-9373-a57afa295d85"))))
        {
            //TenantInfo.Current.Id is "a745a68e-d316-4828-9373-a57afa295d85"
            using (TenantInfo.Change(null))
            {
                //TenantInfo.Current is null!
            }
            //TenantInfo.Current.Id is "a745a68e-d316-4828-9373-a57afa295d85"
        }
    }
}
```

Change Current Tenant

```
public static IDisposable Change(TenantInfo tenantInfo)
{
    var oldValue = Current;
    Current = tenantInfo;
    return new DisposeAction(() =>
    {
        Current = oldValue;
    });
}
```

```
public class DisposeAction : IDisposable
{
    private readonly Action _action;

    public DisposeAction(Action action)
    {
        _action = action;
    }

    public void Dispose()
    {
        _action();
    }
}
```

Multi-Tenancy Middleware Implementation

```
public async Task Invoke(HttpContext httpContext)
{
    using (TenantInfo.Change(FindTenant(httpContext)))
    {
        await _next(httpContext);
    }
}

private TenantInfo FindTenant(HttpContext httpContext)
{
    var tenantId = FindFromClaims(httpContext) ??
                   FindFromDomain(httpContext) ??
                   FindFromHeader(httpContext) ??
                   FindFromCookie(httpContext);

    if (tenantId == null)
    {
        return null;
    }

    return new TenantInfo(Guid.Parse(tenantId));
}
```

ASP.NET Core Multi-Tenancy Middleware

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
        services.AddAuthentication().AddCookie();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseAuthentication();

        app.UseMiddleware<MultiTenancyMiddleware>();

        app.UseMvcWithDefaultRoute();
    }
}
```

Change Tenant in a Background Job

```
public class EmailSendJob : BackgroundJob<EmailSendJobArgs>
{
    private readonly IUserRepository _userRepository;
    private readonly IEmailSender _emailSender;

    public EmailSendJob(
        IUserRepository userRepository,
        IEmailSender emailSender)
    {
        _userRepository = userRepository;
        _emailSender = emailSender;
    }

    public override void Execute(EmailSendJobArgs args)
    {
        using (TenantInfo.Change(new TenantInfo(args.TenantId)))
        {
            var user = _userRepository.GetById(args.UserId);

            _emailSender.Send(
                user.EmailAddress,
                args.EmailTitle,
                args.EmailBody
            );
        }
    }
}
```

Summary

- Determine the current tenant from the current HTTP request
- Use ambient context pattern to set, change and get the current tenant
- Create an ASP.NET Core middleware to set Tenant Id before any action execution
- Explicitly set current tenant in a background job

Multi-Tenancy

Database / Data Isolation

Dynamically Select the Connection String

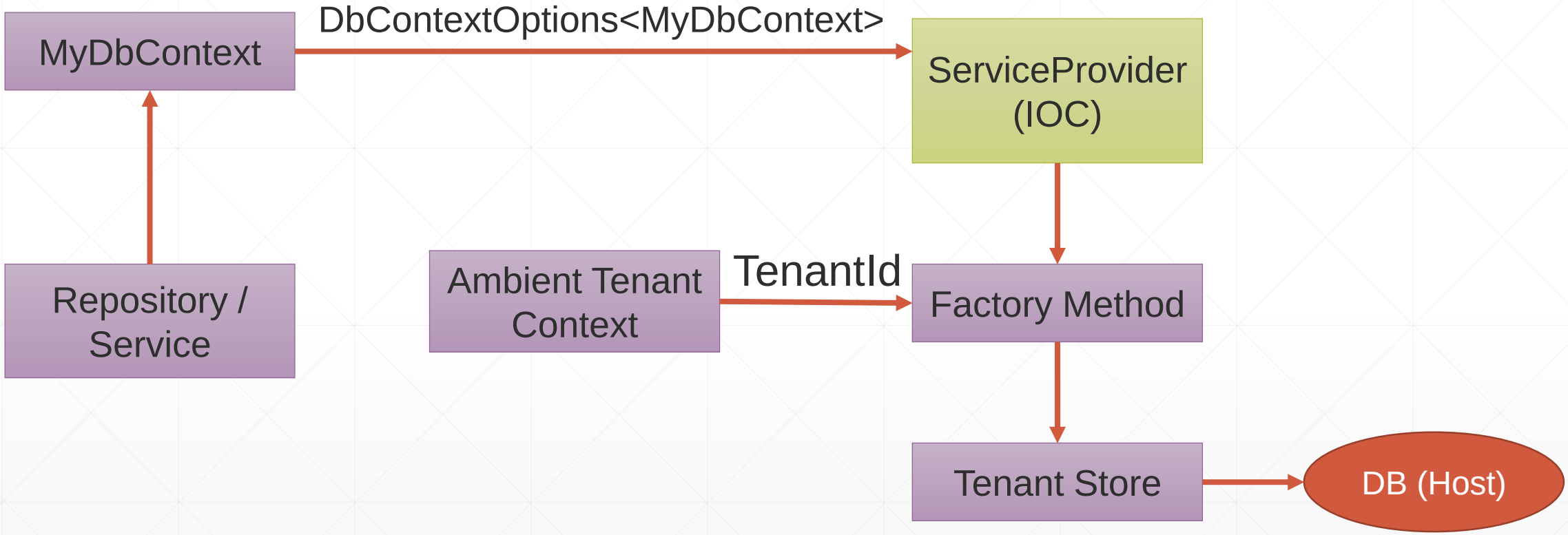
```
services.AddDbContext<TDbContext>(options =>
{
    options.UseSqlServer("Server=localhost;Database=MultiTenancyDraft...");
});
```

```
services.AddMultiTenantDbContext<MyDbContext>((options, connString) =>
{
    options.UseSqlServer(connString);
});
```

Dynamically Select the Connection String

```
public class MyDbContext : DbContext
{
    public MyDbContext(DbContextOptions<MyDbContext> options)
        : base(options)
    {
    }
}
```

Dynamically Select the Connection String



Dynamically Select the Connection String

```
public static void AddMultiTenantDbContext<TDbContext>(
    this IServiceCollection services,
    Action<DbContextOptionsBuilder<TDbContext>, string> optionsBuilder)
    where TDbContext : DbContext
{
    services.AddDbContext<TDbContext>();

    services.AddTransient(serviceProvider =>
    {
        var connString = GetConnectionString(serviceProvider);

        var builder = new DbContextOptionsBuilder<TDbContext>();
        optionsBuilder(builder, connString);
        return builder.Options;
    });
}
```

Dynamically Select the Connection String

```
private static string GetConnectionString(IServiceProvider serviceProvider)
{
    if (TenantInfo.Current == null)
    {
        return GetDefaultConnectionString();
    }

    var tenantStore = serviceProvider.GetRequiredService<ITenantStore>();
    return tenantStore.FindConnectionString(TenantInfo.Current.Id) ??
           GetDefaultConnectionString();
}
```

```
public static string GetDefaultConnectionString()
{
    return "Server=localhost;Database=MultiTenancyDraft...";
}
```

Data Filtering: Manual Way

```
var currentTenantId = TenantInfo.Current?.Id;
var products = _myDbContext.Products
    .Where(p => p.TenantId == currentTenantId)
    .ToList();
```

Automatic Data Filtering: Repository Pattern

```
public class ProductRepository
{
    private readonly MyDbContext _dbContext;

    public ProductRepository(MyDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public IQueryable<Product> GetAll()
    {
        var currentTenantId = TenantInfo.Current?.Id;
        return _dbContext.Products.Where(p => p.TenantId == currentTenantId);
    }

    public Product FindByName(string name)
    {
        return GetAll().FirstOrDefault(p => p.Name == name);
    }
}
```

Automatic Data Filtering: Repository Pattern

- Pros
  - **Easy to implement**
  - **ORM Independent**
- Cons
  - Limited – **Can be bypassed** by directly using DbContext
  - Limited – Does not work for **navigation properties**
  - Open to leak (Repository developer may forget it)

Automatic Data Filtering: EF Core Global Filters

```
public class MyDbContext : DbContext
{
    public Guid? CurrentTenantId => TenantInfo.Current?.Id;

    public DbSet<Product> Products { get; set; }

    public MyDbContext(DbContextOptions<MyDbContext> options)
        : base(options)
    {

    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Product>(b =>
        {
            b.HasQueryFilter(p => p.TenantId == CurrentTenantId);
        });
    }
}
```

Automatic Data Filtering: EF Core Global Filters

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    foreach (var entityType in modelBuilder.Model.GetEntityTypes())
    {
        ConfigureGlobalFiltersMethodInfo
            .MakeGenericMethod(entityType.ClrType)
            .Invoke(this, new object[] { modelBuilder, entityType });
    }
}
```

Automatic Data Filtering: EF Core Global Filters

```
protected void ConfigureGlobalFilters<TEntity>(ModelBuilder modelBuilder, IMutableEntityType entityType)
    where TEntity : class
{
    if (entityType.BaseType != null)
    {
        return;
    }

    var filterExpression = CreateFilterExpression<TEntity>();
    if (filterExpression != null)
    {
        modelBuilder.Entity<TEntity>().HasQueryFilter(filterExpression);
    }
}
```

Automatic Data Filtering: EF Core Global Filters

```
protected virtual Expression<Func<TEntity, bool>> CreateFilterExpression<TEntity>()
    where TEntity : class
{
    Expression<Func<TEntity, bool>> expression = null;

    if (typeof(ISoftDelete).IsAssignableFrom(typeof(TEntity)))
    {
        expression =
            e => !((ISoftDelete)e).IsDeleted || ((ISoftDelete)e).IsDeleted != IsSoftDeleteFilterEnabled;
    }

    if (typeof(IMultiTenant).IsAssignableFrom(typeof(TEntity)))
    {
        Expression<Func<TEntity, bool>> multiTenantFilter =
            e => ((IMultiTenant)e).TenantId == CurrentTenantId ||
                 (((IMultiTenant)e).TenantId == CurrentTenantId) == IsMultiTenantFilterEnabled;

        expression = expression == null
            ? multiTenantFilter
            : CombineExpressions(expression, multiTenantFilter);
    }

    return expression;
}
```

Automatic Data Filtering: EF Core Global Filters

- Pros
  - **Natively** works with EF Core
  - Supports **navigation properties** as well
- Cons
  - Limited – Does not work if you directly work with **SQL, stored procedures**… etc.

Automatic Data Filtering: Other Options

- **Row Level Security** – Available for SQL Server and Azure SQL Database
  - Pros: Completely integrated to DBMS, **works for everything**
  - Cons: Relatively **complex** to implement, specific to DBMS
- **Azure Elastic Database Pool**
  - Pros: **Easy to scale**
  - Cons: Only for ***db per tenant*** scenario, vendor dependency

Automatic Data Filtering: Row Level Security

Set **Session Context** while changing the current tenant:

```
// Set TenantId in SESSION_CONTEXT to shardingKey
// to enable Row-Level Security filtering.
SqlCommand cmd = conn.CreateCommand();
cmd.CommandText =
    @"exec sp_set_session_context
        @key=N'TenantId', @value=@shardingKey";
cmd.Parameters.AddWithValue("@shardingKey", tenantId);
cmd.ExecuteNonQuery();
```

Automatic Data Filtering: Row Level Security

Create a **policy** to filter/block data

```
CREATE FUNCTION rls.fn_tenantAccessPredicate(@TenantId int)
    RETURNS TABLE
    WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_accessResult
        -- Use the user in your application's connection string.
        -- Here we use 'dbo' only for demo purposes!
        WHERE DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('dbo')
        AND CAST(SESSION_CONTEXT(N'TenantId') AS int) = @TenantId;
GO

CREATE SECURITY POLICY rls.tenantAccessPolicy
    ADD FILTER PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Blogs,
    ADD BLOCK  PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Blogs,
    ADD FILTER PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Posts,
    ADD BLOCK  PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Posts;
GO
```

Automatic Data Filtering: Row Level Security

Automatically set TenantId for new entities

```
-- Create default constraints to auto-populate TenantId with the
-- value of SESSION_CONTEXT for inserts.
ALTER TABLE Blogs
    ADD CONSTRAINT df_TenantId_Blogs
    DEFAULT CAST(SESSION_CONTEXT(N'TenantId') AS int) FOR TenantId;
GO
```

Setting Tenant Id for New Entities

- In DbContext.SaveChanges()
  - Get new entities for change tracker, find IMultiTenant entities, set current TenantId
  - Problem: May set wrong tenantId

```
public class TenantChangeDemo3
{
    public void CreateProduct(Guid tenantId, string name)
    {
        using (TenantInfo.Change(new TenantInfo(tenantId)))
        {
            //TODO: ProductRepository.Insert(new Product(name))
        }

        //TODO: SaveChanges() //Current Tenant is not "tenant1"!
    }
}
```

Setting Tenant Id for New Entities

```
public class Product : IMultiTenant
{
    public Guid Id { get; set; }

    public Guid? TenantId { get; set; }

    public string Name { get; set; }

    protected Product()
    {

    }

    public Product(string name, Guid? tenantId = null)
    {
        Id = Guid.NewGuid();
        Name = name;
        TenantId = tenantId;
    }
}
```

Safe Way to Manipulate Data

```
public class ProductService
{
    private readonly ProductRepository _productRepository;

    public ProductService(ProductRepository productRepository)
    {
        _productRepository = productRepository;
    }

    public void ChangeName(Guid productId, string newName)
    {
        var product = _productRepository.FindById(productId);
        product.Name = newName;
        _productRepository.Update(product);
    }
}
```

Summary

- Dynamically set connection string based on the current tenant for a **database per tenant** approach: Register as a factory method
- Automatically filter data based on the current tenant for a **shared database** approach.
  - Manual way, Repository pattern, EF Core Global Query Filters, SQL Server Row Level Security (RLS)
- Best practice to set Tenant Id for new entities: Set on constructor
- Safe way to update an existing entity: Query first

Multi-Tenancy

Enable/Disable

Disable/Enable By Scope

- May need to query on all tenants
- Can be implemented using ambient context pattern
- Problem: Not easy for multi-database scenario

```
List<Person> people;

using (_multiTenantFilter.Disable())
{
    //Filter disabled manually
    people = _personRepository.ToList();
    people.Count.ShouldBe(3);
}

//Filter re-enabled automatically
people = _personRepository.ToList();
people.Count.ShouldBe(1);
```

Multi-Tenancy

Database Migrations

Schema / Data Migration

- Problem for multiple-databases
- Solution: Upgrade **all in one** with a custom tool
  - Pros: Easy to implement. All tenants are in the same version
  - Cons: May get too long time for big number of tenants and data. All tenants wait for all upgrade progress
- Solution: Upgrade the **application servers immediately**, upgrade **databases individually** on first access
  - Pros: Upgrading is distributed to time. A tenant does not wait for another
  - Cons: First accessing user may wait too much. Even we get timeout exception. Also, we don’t control the upgrade speed

Schema / Data Migration

- Solution: Multiple versions concurrently
  - Split the application servers into two parts: Upgraded tenants use the new application, other tenants use the old application
- Pros:
  - Minimum waiting for every tenant
  - Upgrading may be scheduled for every individual tenant and they can be informed
  - Allows us to perform A/B tests and previews
- Cons
  - Requires multiple application servers – but reasonable for a big system
  - Harder to implement, maintain and monitor

Multi-Tenancy

SaaS Features

Feature / Package / Subscription System

- **Define features** of the application. Feature Types:
  - **On/Off**: Excel export, Replying by email (for a support app)
  - **Numeric**:10 users, 20,000 emails/month
  - **Selection**: one of the available options
- Group features into **packages/editions**
- **Subscribe** packages by tenants
- **Check features**: Declarative or by code

Feature Checking

```
[RequiresFeature("ExportToExcel")]
public async Task<FileDto> GetReportToExcel(...)
{
    ...
}
```

```
public async Task<FileDto> GetReportToExcel(...)
{
    if (await FeatureChecker.IsEnabledAsync("ExportToExcel"))
    {
        throw new AbpAuthorizationException("You don't have this feature: ExportToExcel");
    }

    ...
}
```

Declarative Feature Check
Implementation Options

- MVC Action Filters
  - Easy to implement. Naturally works within ASP.NET Core
  - Limited to Controller actions
- Method Interception using dynamic proxying (interception)
  - Works everywhere
  - Limited to virtual methods
- Weaving: Mono.Cecil, Fody, Postsharp… etc.

Declarative Feature Check: Interception

```
public class RequiresFeatureInterceptor : IInterceptor
{
    public void Intercept(IInvocation invocation)
    {
        var featureAttributes = invocation.MethodInvocationTarget
            .GetCustomAttributes()
            .OfType<RequiresFeatureAttribute>();

        foreach (var featureAttribute in featureAttributes)
        {
            foreach (string featureName in featureAttribute.Features)
            {
                if (IsEnabled(featureName) == false)
                {
                    throw new AbpAuthorizationException(
                        "Required feature was not satisfied: " + featureName
                    );
                }
            }
        }

        invocation.Proceed();
    }

    private bool IsEnabled(string featureName)
    {
        throw new System.NotImplementedException();
    }
}
```

Multi-Tenancy

Summary

Summary

- What is SaaS & Multi-Tenancy? Tenant v.s. Host
  - Benefits: Maximum Utilization, easy to maintain
  - Difficulties: Isolation, Performance, Customization, Security, Backup (per tenant)
- Database Options: DB per tenant, single DB, Hybrid
- Ideal Multi-Tenant & Stateless Application Design
- Determine the current tenant
  - Sources to obtain current tenant (HTTP request)
  - Ambient context pattern & AsyncLocal
- Change the current tenant

Summary

- ASP.NET Core Multi-tenancy Middleware
- Dynamically Select the Connection String
- Automatically Filtering Tenant Data
  - EF Core Global Query Filters
  - SQL Server Row Level Security
- Safe way to insert and update entities
- Disable/Enable multi-tenancy
- Database migrations
- SaaS Features & Dynamic Proxying

Github @hikalkan
Twitter @hibrahimkalkan
Email halil.kalkan@volosoft.com

Slide Source: *https://github.com/hikalkan/presentations/tree/master/2018-11-22-Multi-Tenancy*

Thank you…

Halil İbrahim Kalkan