Provision and Catalog

A SaaS-in-a-Box Tutorial

# Purpose of this tutorial

The Introduction to the WTP SaaS application tutorial showed how the WTP app provisions a tenant database with its initial schema and register it in the catalog, making it available for use by the rest of the application. To do this, the app uses ARM APIs and SQL commands. The same APIs and commands are available for scripting also. This tutorial explores using PowerShell to do the same thing, and uses these scripts as a vehicle to explore the provisioning and cataloging patterns used, and how the catalog has been implemented.

# Introduction to the SaaS Catalog Pattern

In a database-backed multi-tenant SaaS application, it’s important to know where information for a tenant is stored. In the SaaS Catalog pattern, a catalog database is used to hold a mapping between a tenant and a database in which that tenant’s data is stored. The basic pattern applies whether a multi-tenant or a single-tenant database is used.



The catalog in the WTP app is implemented using the Shard Management technology in the Elastic Database Client Library (EDCL). EDCL is responsible for creating and managing a database-backed ‘catalog’ in which a ‘shard map’ is maintained, which contains the mapping between keys and databases ("shards"). While the mapping data is readily accessible in the catalog database, *it should not be manipulated except through the APIs exposed by the EDCL*. Directly manipulating the data risks corrupting the catalog and is not supported.

EDCL also provides important features that enable a client application to connect to and manipulate the catalog and to use it to retrieve an ADO.NET connection for a given key value, enabling the application to connect to the correct database. The client caches this connection information to minimize the traffic to the catalog database and speed up the application.

# Using the Catalog

The Wingtip application is built using a single-tenant database model. A unique tenant key is used for each tenant and used to map the tenant to its database. Exactly how the key is formed is not critical, although the key either must be known to, or derivable by, any client application that needs to access the tenant’s data. In the WTP application, the key is formed from a hash of the tenant’s name. This allows the tenant name portion of the application URL to be used to construct the key and retrieve the connection. Other id schemes could be used without impacting the overall pattern.

This tutorial explores how the process used when provisioning a tenant.

# Setting Up

The **SaaS in a Box Learning Modules and Utilities.zip** should be downloaded and extracted to a convenient folder.

The **WTP Application** must be deployed and setup, with the catalog initialized from the Demo Assistant app. See the Introduction to the WTP SaaS Application tutorial.

**PowerShell ISE** is recommended for executing scripts and following their execution in debug mode.

**PowerShell Tips**

* Open and configure demo- scripts in the PowerShell ISE.
* Use F5 to run the script (using F8 is not advised as the $PSScriptRoot variable is not evaluated when running snippets of a script).
* Use F9 to set a breakpoint to let you trace the script in debug mode to see how it works
* Use F10 to step through the script, F11 to step into a function, and Shift-F11 to step out.

**SSMS** can be used to explore database schema and execute SQL queries directly.

# Walkthrough

## Getting Started

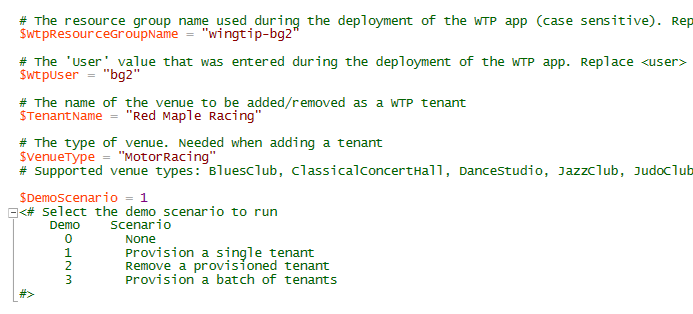
1. Update the user configuration file used by all tutorial scripts. Update again if you redeploy the app.
   1. Open ...\Learning Modules\UserConfig.psm1 in **PowerShell ISE**
   2. Modify **$userConfig.ResourceGroupName** to the resource group used for the deployed app.
   3. Modify **$userConfig.Name** to the User name used for the deployed app.
2. Open the following scripts and modules in **PowerShell ISE**
   1. …\Learning Modules\Provision and Catalog\Demo-ProvisionAndCatalog.ps1
   2. …\Learning Modules\Provision and Catalog\New-Tenant.ps1,
   3. …\Learning Modules\Provision and Catalog\New-TenantBatch.ps1
   4. …\Learning Modules\Common\CatalogAndDatabaseManagement.psm1
   5. …\Learning Modules\Common\ShardManagement.psm1

## Exercise 1: Provision a new tenant, from 40,000 ft

This exercise will quickly create a new tenant and register it in the catalog.

1. In …\**Demo-ProvisionAndCatalog**.ps1
2. Set **$TenantName** to the name of the Venue you plan to provision
3. Set **$VenueType** to one of the pre-defined venue types. You’ll add more in another tutorial.
4. Set **$DemoScenario = 1**, **Provision a single tenant**
5. Execute using F5

Image below shows how the script might look before you execute it with F5

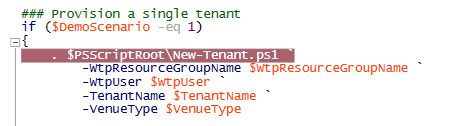


The script will execute and create a new tenant – provisioning a new database, initializing it and registering it in the catalog. It will then open both the events app (tenant user) and the admin app on the new tenant. Note the URL patterns used for both apps.

## Exercise 2: Provision a new Tenant from 500 ft

Now do the same again and this time step through the script and see what is done.

1. With the same script, set **$TenantName** to a new venue name and set **$VenueType** to a different value.
2. Before executing, use F9 to add a breakpoint to the New-Tenant.ps1 execution line



1. Use F5 to execute, and then F11 to step into New-Tenant.ps1.
2. Trace its execution using F10 and F11 to step into the functions it calls.
3. First it imports modules containing useful PowerShell functions:



* 1. SubscriptionManagement manages Azure login and subscription selection.
  2. CatalogAndDatabaseManagement provides a catalog- and tenant-level abstraction over the Shard Management functions. This is a key module which encapsulates much of the catalog pattern and is worth exploring at you leisure.

1. **Get the configuration**. Step into Get-Configuration with F11 to see how config is specified. All the standard names of resources are defined here. It’s not recommended you change any of these values until you are familiar with the scripts!

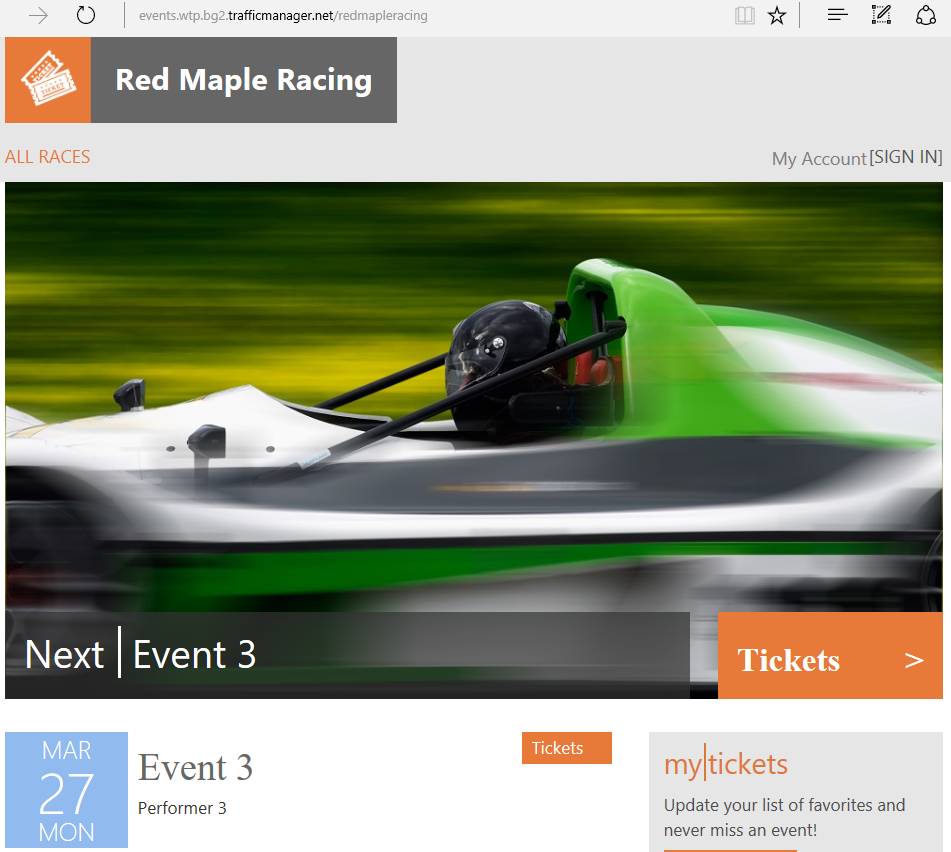
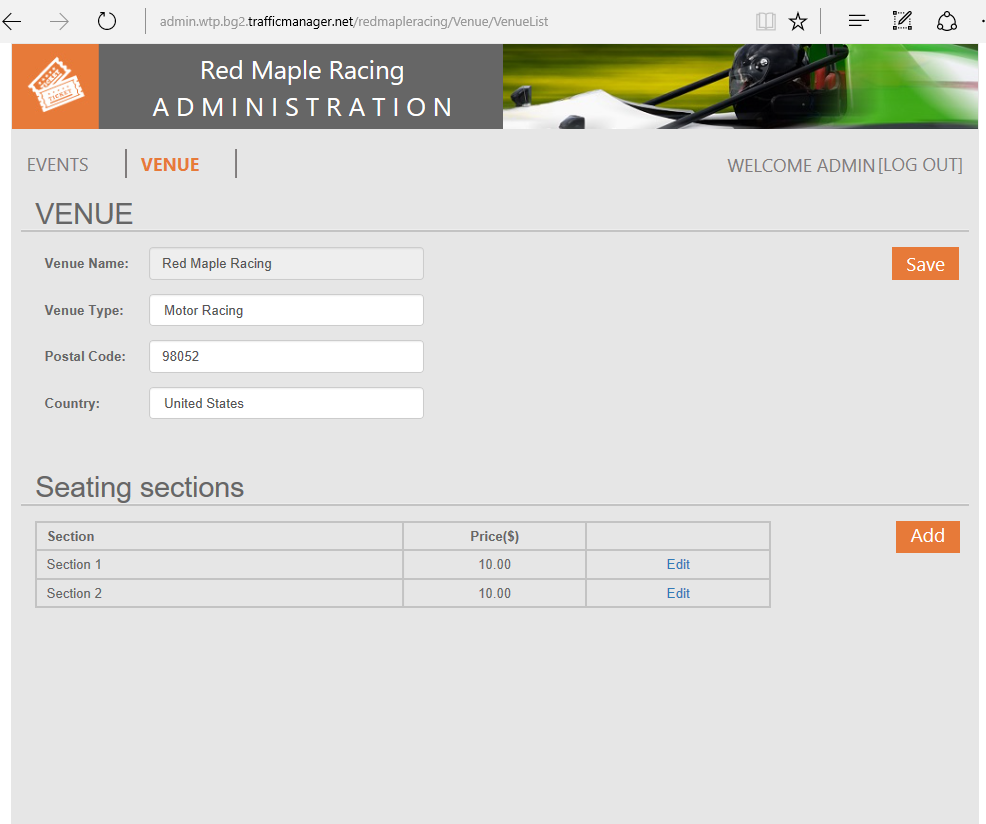


1. **Get a catalog object**. Step into Get-Catalog to see how the catalog is initialized using Shard Management functions that are imported from AzureShardManagement.psm1



* 1. $catalogServerName is constructed using the standard stem plus your User name.
  2. $catalogDatabaseName is retrieved from config – “customercatalog”.
  3. $shardMapManager object is initialized from the catalog database.
  4. $shardMap object is initialized from the “customercatalog” shard map in the catalog database.
  5. A catalog object is composed and returned, and used in the higher-level script.

1. **Get the tenant key**. A hash function is used to create the tenant key from the tenant name.
2. **Check the tenant doesn’t already exist**. The catalog is checked to ensure the key is new.
3. **The tenant database is provisioned with New-TenantDatabase.** Use F11 to see how the database is provisioned using an ARM (Azure Resource Manager) template.
   1. The database name is constructed from the tenant name to make it clear what’s going on. Other strategies for database naming could easily be used.
   2. Two ARM templates are supported (which is used is controlled by a config setting).
      1. **Create a new database by copying the ‘golden’ database** (baseTenantDB) on the catalog server), or
      2. **Create a new empty database and then import a ‘golden’ bacpac** exported from baseTenantDB.
   3. The WTP app currently **provisions by copy** so the scripts use the ARM copy-based template as the default.
   4. The ARM templates are in …\Learning Modules\Common\
      1. tenantdatabasetemplate.json
      2. tenantdatabasecopytemplate.json
   5. Once the ARM template is configured, it is submitted to ARM, which manages its deployment with the resource providers – in this case, the SQL Database service.
   6. The **tenant database is initialized** with the venue (tenant) name and the venue type. Other initialization could also be done at this point.
4. **The tenant database is registered in the catalog** **with Add-TenantDatabaseToCatalog** using the tenant key. Use F11 to step into the function:
   1. The catalog database is added to the shard map (the list of known databases).
   2. The mapping that links the key value to the shard is created.
   3. Add additional meta data to describe the tenant.
5. Back in the original demo- script the **admin** and **events** pages are then opened in the browser. You can see by the background photo used how the default venue type has been overridden by the venue type selection you made in the script.

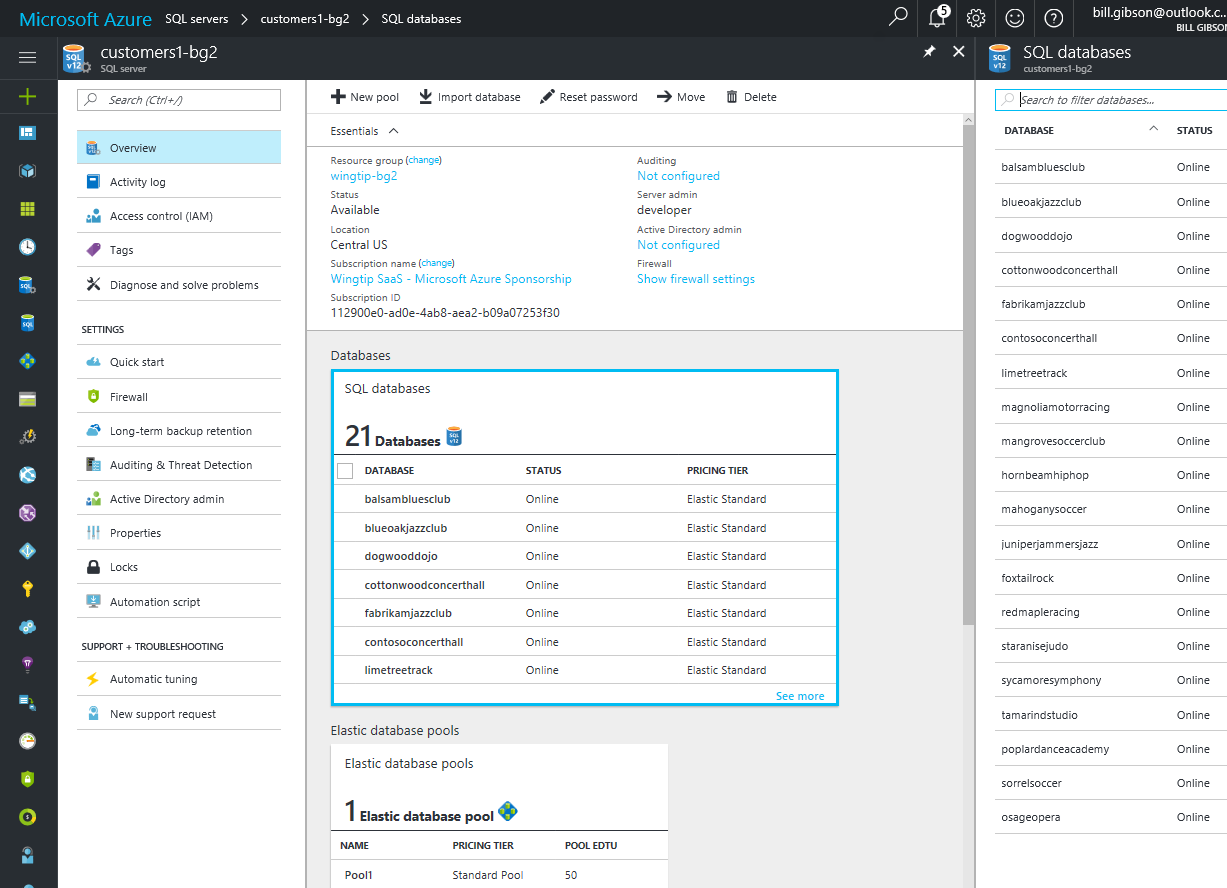
## Exercise 3: Provision a batch of tenants

This exercise provisions a batch of tenants in a single action. It uses an ARM template that controls the batch and then delegates provisioning of each database to a linked template. Using templates like this allows ARM to broker the provisioning process for your script, provisioning databases in parallel where it can and handling any retries if needed, so optimizing the overall process.

Provisioning takes a few minutes, and as it is needed for the Performance Monitoring and Management tutorial, it’s recommended you do this now if you plan to do that tutorial in the same session.

1. In …\**Demo-ProvisionAndCatalog**.ps1 in PowerShell ISE.
2. Set **$DemoScenario = 3,** **Provision a batch of tenants**.
3. Execute the script using **F5**.

The script will deploy a batch of 17 additional tenants. The script is idempotent, so it can be run again should it be interrupted for any reason. After the batch has deployed the script will open the server blade, which if you click on the SQL Databases list part, will look like the image below.



## Other provisioning patterns

Other provisioning patterns not included in this tutorial include:

**Pre-provisioning databases.** This pattern exploits the fact that databases in a pool are not charged for (the pool is charged for, not the databases), and that idle databases consume no resources. By pre-provisioning databases in a pool and then assigning them when needed, tenant onboarding time can be cut dramatically from a few minutes to a few seconds. The number of databases pre-provisioned could be adjusted as needed to keep a buffer suitable for the anticipated provisioning rate.

**Auto-provisioning.** In this pattern, a dedicated provisioning service is used to provision servers, pools and databases automatically as needed – including pre-provisioning databases in pools if desired. And if databases are de-commissioned and deleted, gaps in pools can be filled by the provisioning service as desired. Such a service could be simple or complex – for example, handling provisioning across multiple geographies, and could set up geo-replication automatically if that strategy is being used for DR. With this pattern, client application or scripts would submit a provisioning request to a queue to be processed by the provisioning service and would poll the service to determine completion. If pre-provisioning is used requests would be handled very quickly with the service managing provisioning of a replacement databases in the background.

## Clean up

If you don’t plan to continue with another tutorial, it’s recommended you delete all the resources to save money. Simply delete the resource group in which the WTP application was deployed and all resources created during this and all the other SaaS in a Box tutorials will be deleted.

## Resources

For more information on the elastic data client library see:

<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-database-client-library>