Building & Deploying a Microservice using Platform Management APIs

Created by Pascal Enz (Group), last modified on 03 Apr 2019

Clean up
Using the Platform Management API creates actual infrastructure resources. Please always clean up after yourself and delete any test resources immediately.

Sample Solution
You can find a fully working sample solution here:
https://scm.dimensiondata.com/service-layer/samples/my-to-do-list

Microservice Template
You can find a template for creating a Platform managed Microservice here:

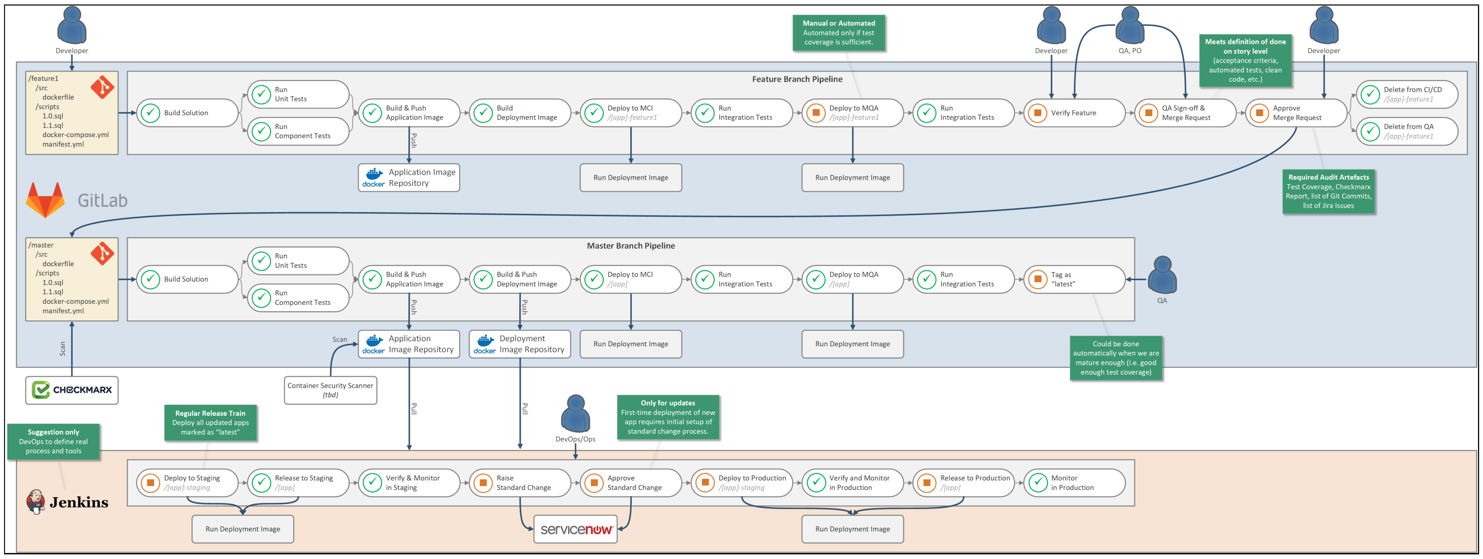
https://scm.dimensiondata.com/service-layer/common-services/MicroserviceTemplate

Overview

Contents

Proposed Pipeline

The steps on this page are based on the proposed pipeline below.

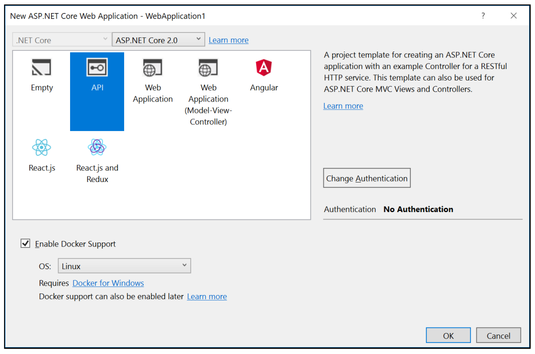

Developer
/feature1
/src
dockerfile
/scripts
1.0.sql
1.1.sql
docker-compose.yml
manifest.yml
Feature Branch Pipeline
Build Solution
Run Unit Tests
Run Component Tests
Build & Push Application Image
Build Deployment Image
Deploy to MCI /[app]-feature1
Run Integration Tests
Deploy to MQA /[app]-feature1
Run Integration Tests
Verify Feature
QA Sign-off & Merge Request
Approve Merge Request
Delete from CI/CD /[app]-feature1
Delete from QA /[app]-feature1
Manual or Automated
Automated only if test coverage is sufficient.
Developer
QA, PO
Developer
Meets definition of done on story level (acceptance criteria, automated tests, clean code, etc.)
Push
Application Image Repository
Run Deployment Image
Run Deployment Image
Required Audit Artefacts
Test Coverage, Checkmarx Report, list of Git Commits, list of Jira Issues
GitLab
/master
/src
dockerfile
/scripts
1.0.sql
1.1.sql
docker-compose.yml
manifest.yml
Master Branch Pipeline
Build Solution
Run Unit Tests
Run Component Tests
Build & Push Application Image
Build & Push Deployment Image
Deploy to MCI /[app]
Run Integration Tests
Deploy to MQA /[app]
Run Integration Tests
Tag as "latest"
QA
Push
Push
Scan
Application Image Repository
Deployment Image Repository
Run Deployment Image
Run Deployment Image
Could be done automatically when we are mature enough (i.e. good enough test coverage)
Scan
CHECKMARX
Container Security Scanner (tbd)
Pull
Pull
DevOps/Ops
Regular Release Train
Deploy all updated apps marked as "latest"
Only for updates
First-time deployment of new app requires initial setup of standard change process.
Suggestion only
DevOps to define real process and tools
Jenkins
Deploy to Staging /[app]-staging
Release to Staging /[app]
Verify & Monitor in Staging
Raise Standard Change
Approve Standard Change
Deploy to Production /[app]-staging
Verify and Monitor in Production
Release to Production /[app]
Monitor in Production
Run Deployment Image
servicenow
Run Deployment Image

Step 1: Develop an App

Microservices can be build with any language as long as they are packaged into Docker containers.

Develop a .NET Core App

In the latest version of Visual Studio 2017, create a new ASP.NET Core 2.x Web Application.


New ASP.NET Core Web Application - WebApplication1
Empty
API
Web Application
Web Application (Model-View-Controller)
Angular
React.js
React.js and Redux
Change Authentication
OK
Cancel

.NET Core / ASP.NET Core 2.0 Learn more

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
Learn more

Authentication **No Authentication**

☑ Enable Docker Support

OS: Linux

Requires Docker for Windows

Docker support can also be enabled later Learn more

App Configuration

Apps can retrieve their settings, secrets, and credentials from the Platform Management REST API . However, to simplify the integration and hide the API from the app code, the *DimensionData.ServiceLayer.Sdk* Nuget package will provide a range of .NET Standard configuration providers which seamlessly load the configuration into the app. It can be configured at startup time as seen below.

› Sample App Configuration

```
public static class Program
{
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((context, builder) =>
            {
                builder.AddPlatformManagement("DEMO1")
                    .AddAppSettings() // Consul KV
                    .AddAppSecrets() // Vault KV
                    .AddSqlDatabaseConnectionString("MyDemoDatabase_V1", options => options.ConnectionStringName = "MyDemoDatabase") //
                    .AddVaultMountConnectionInfo("MyDemoVaultMount1", options => options.ConfigurationPath = "VaultMount:MyDemoVaultMoun
                    .AddVaultMountConnectionInfo("MyDemoVaultMount2", options => options.ConfigurationPath = "VaultMount:MyDemoVaultMoun
                    .AddRabbitMQConnectionInfo("sl-global", options => options.ConfigurationPath = "RabbitMq:Global") // Global rabbitmq
                    .AddRabbitMQConnectionInfo("sl", options => options.ConfigurationPath = "RabbitMq:Local"); // Local rabbitmq connect
            })
            .Build();
}
```

Your app can later easily retrieve the various settings, secrets and credentials later thru the IConfiguration interface. Below is an example how to get those values from IConfiguration, bind them to a class and register them as Options, and get database connection string:

› Sample Option Configuration

```
public class MyDemoOption
{
    public string Value1 { get; set; }
    public string Value2 { get; set; }
}

public static void ConfigureOptions(this IServiceCollection services, IConfiguration configuration)
{
    services.Configure<MyDemoOption>(options => configuration.GetSection("MyDemoOption").Bind(options)); // Custom config from Consul/Va
    services.Configure<VaultMountConnectionInfo>("MyDemoVaultMount1", options => configuration.GetSection("VaultMount:MyDemoVaultMount1"
    services.Configure<VaultMountConnectionInfo>("MyDemoVaultMount2", options => configuration.GetSection("VaultMount:MyDemoVaultMount2"
    services.Configure<RabbitMQConnectionInfo>("rabbitmq-global", options => configuration.GetSection("RabbitMq:Global").Bind(options));
    services.Configure<RabbitMQConnectionInfo>("rabbitmq-local", options => configuration.GetSection("RabbitMq:Local").Bind(options));
}

private static void RegisterDataContext(IServiceCollection services, IConfiguration configuration)
{
    // Make sure you get the database connection string from within the IAction when registering the database context, so that it will p
    services.AddDbContext<DomainDataContext>(options =>
        options
            .UseSqlServer(configuration.GetConnectionString("MyDemoDatabase")));
}
```

The options can then later be injected into other classes as IOption/IOptionSnapshot, for example:
› Sample Option Consumption

```
class MyTestClass
{
    private readonly MyTestOption _testOption;
    private readonly VaultMountConnectionInfo _vaultMountInfo1;
    private readonly VaultMountConnectionInfo _vaultMountInfo2;
    private readonly RabbitMQConnectionInfo _rabbitmqGlobalInfo;
    private readonly RabbitMQConnectionInfo _rabbitmqLocalInfo;

    public MyTestClass(
        IOption<MyTestOption> testOption,
        IOptionsSnapshot<VaultMountConnectionInfo> vaultMountInfo,
        IOptionsSnapshot<RabbitMQConnectionInfo> rabbitmqInfo)
    {
        _testOption = testOption.Value
        _vaultMountInfo1 = vaultMountInfo.Get("MyDemoVaultMount1");
        _vaultMountInfo2 = vaultMountInfo.Get("MyDemoVaultMount2");
        _rabbitmqGlobalInfo = rabbitmqInfo.Get("rabbitmq-global");
        _rabbitmqLocalInfo = rabbitmqInfo.Get("rabbitmq-local");
    }
}
```

IOption vs IOptionSnapshot
You need to use IOptionSnapshot for options that will change over time, such as the rotating credentials used by database connection string, vault mount credentials, and perhaps even app settings and secrets. By using IOptionSnapshot, it will cause it to reload the options if necessary instead of caching it for the lifetime of the app.

You also need to be careful when using a singleton in dependency injection (such as a RabbitMQ message subscriber), by making sure the IOptionSnapshot values or database connection strings does not get cached. One way to solve this is by creating a new lifetime scope when calling cache sensitive components (such as database context) within the singleton.

See this page from Microsoft on differences between IOption and IOptionSnapshot.

Metrics (Prometheus)

Please see Metrics Collection (Prometheus) for how to enable the metrics collection endpoint using Prometheus.

Containerisation

You then need to setup a dockerfile to create a Docker image that contains the run-time version of your app, for example:
› Sample Dockerfile

```
FROM microsoft/dotnet:2.1-runtime
WORKDIR /app
EXPOSE 80
COPY ./src/TestApplication.WebHost/publish .
ENTRYPOINT ["dotnet", "TestApplication.WebHost.dll"]
```

This dockerfile does the following:

- Use "microsoft/dotnet:2.1-runtime" as the base image, which contain all the required tools to run a .Net Core application
- Create and make "/app" as the working directory
- Expose port 80 from the container (ensure your app host the APIs on port 80 as well)
- Copy your app's run time files from the "publish" folder into the container (you need to have build your app first before build the Docker image)
- Use the "dotnet" CLI to run the app

Develop a Node.js App

https://nodejs.org/en/docs/guides/nodejs-docker-webapp/

💡 You can easily create a Node.js express app in Visual Studio after installing the tools. However, make sure you change the application's listening port from 3000 to something else because that port is already used by the underlying container base image.

Step 2: Build the App

Build and Push the Container Image

Open a command prompt in the root folder of the solution and enter the commands similar to the ones below in order to compile the code, build the container image and upload it to a registry.

```
docker login artifactory.devops.itaas-cloud.com:6553
docker build -t artifactory.devops.itaas-cloud.com:6553/WebApplication1:0.1 -f .\src\dockerfile .
docker push artifactory.devops.itaas-cloud.com:6553/WebApplication1:0.1
docker rmi artifactory.devops.itaas-cloud.com:6553/WebApplication1:0.1
```

Step 3: Create Deployment Files

docker-compose.yml

A *docker-compose.yml* file needs to be provided to deploy the container stack to the container orchestration platform. In should contain at least the following elements.

- The container image location (required)
- The app key under which the service will be exposed on NGNIX (optional)
- Add the health and diagnostics checks (optional)

› Sample: docker-compose.yml

docker-compose.yml

```
version: '2'
services:
  api:
    image: artifactory.devops.itaas-cloud.com:6553/WebApplication1:0.1
```

SQL Scripts

If the app uses a SQL database, schema deployment scripts can be executed via Platform Management API too. However, you must ensure that all scripts referenced in configuration templates are idempotent. Below is an example of idempotent, sequential scripts.

› 1.0.sql

```
IF (SCHEMA_ID('Deployment') IS NOT NULL)
BEGIN
    SET NOEXEC ON
END
GO

CREATE TABLE [dbo].[Table1] (
    [Id]           UNIQUEIDENTIFIER NOT NULL,
    [Name]         NVARCHAR (MAX)   NULL,
    CONSTRAINT [PK.Table1] PRIMARY KEY CLUSTERED ([Id] ASC)
);
GO

CREATE SCHEMA [Deployment] AUTHORIZATION [dbo];
GO

CREATE TABLE [Deployment].[VersionHistory] (
    [MajorVersion] INT           NOT NULL,
    [MinorVersion] INT           NOT NULL,
    [Date]         DATETIME2 (7) NOT NULL,
    PRIMARY KEY CLUSTERED ([MinorVersion] ASC, [MajorVersion] ASC)
);
GO
```

```
CREATE PROCEDURE [Deployment].[CheckVersion]
    @MajorVersion INT,
    @MinorVersion INT
AS
BEGIN
    DECLARE @exists INT

    SELECT @exists = COUNT(*)
        FROM [Deployment].[VersionHistory]
        WHERE ([MajorVersion] = @MajorVersion AND [MinorVersion] >= @MinorVersion) OR ([MajorVersion] > @MajorVersion)

    RETURN @exists
END
GO

CREATE PROCEDURE [Deployment].[RegisterVersion]
    @MajorVersion INT,
    @MinorVersion INT
AS
BEGIN
    INSERT INTO [Deployment].[VersionHistory] ([MajorVersion], [MinorVersion], [Date])
        VALUES (@MajorVersion, @MinorVersion, GETUTCDATE())
END
GO

EXEC [Deployment].[RegisterVersion] @MajorVersion = 1, @MinorVersion = 0;
GO
```

› 1.1.sql

```
DECLARE @exists INT
EXEC @exists = [Deployment].[CheckVersion] @MajorVersion = 1, @MinorVersion = 1
IF (@exists > 0)
BEGIN
    SET NOEXEC ON
END
GO

CREATE TABLE [dbo].[Table2] (
    [Id]           UNIQUEIDENTIFIER NOT NULL,
    [Name]         NVARCHAR (MAX)   NULL,
    CONSTRAINT [PK.Table2] PRIMARY KEY CLUSTERED ([Id] ASC)
);
GO

EXEC [Deployment].[RegisterVersion] @MajorVersion = 1, @MinorVersion = 1
GO
```

Configuration Template (Application Manifest)

Microservices including their databases and containers can be deployed using the Platform Management REST API. However, we strongly recommend writing a Configuration Template to automate the process. Below is an example in the two supported formats, JSON and YAML.

› Sample: manifest.yml (without database)

```
parameters:
- name: SERVICE_PROVIDER_ID
  description: The identifier of the service provider that owns the app.
  type: string
  required: true

variables:
- name: APP_KEY
  value: DEMO1
- name: APP_NAME
  value: Demo Application 1

resources:
- resourceType: App
```

```
  apiVersion: '1.0'
  apiParameters:
    serviceProviderId: "[parameters('SERVICE_PROVIDER_ID')]"
  properties:
    appKey: "[variables('APP_KEY')]"
    name: "[variables('APP_NAME')]"

- resourceType: ContainerStack
  resourceId: CONTAINER_STACK
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    stackName: staging
  properties:
    dockerCompose: "[file('docker-compose.yml')]"
    instances: 2
    publicEndpoint:
      serviceName: api
      healthCheck:
        http: "/health"
        interval: '00:00:15'
        timeout: '00:00:10'
  wait:
    until: "[resources('CONTAINER_STACK').isHealthy]"
    interval: '00:00:05'
    timeout: '00:05:00'

- resourceType: ContainerStack/Release
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    stackName: staging
  properties:
    deleteExisting: true
```

› Sample: manifest.yml (with app settings, app secrets, vault mount and database)

```
parameters:
- name: SERVICE_PROVIDER_ID
  description: The identifier of the service provider that owns the app.
  type: string
  required: true
- name: GIT_BRANCH
  description: The name of the Git branch that triggered the deployment.
  type: string
  required: true
  defaultValue: master

variables:
- name: APP_KEY
  value: DEMO1
- name: APP_NAME
  value: Demo Application 1
- name: DATABASE_NAME
  value: MyDemoDatabase_V1
- name: CREDENTIAL_ID
  value: "[newGuid()]"
- name: CREDENTIAL_SECRET
  value: "[newSecret(30)]"

resources:
- resourceType: ServicePrincipal
  resourceId: SERVICE_PRINCIPAL
  apiVersion: '1.3'
  apiParameters:
    organizationId: "[parameters('SERVICE_PROVIDER_ID')]"
  properties:
    spn: "[concat(variables('APP_KEY'), '_', parameters('GIT_BRANCH'))]"
    name: "[concat(variables('APP_KEY'), ' ', parameters('GIT_BRANCH'), ' Service Principal')]"
    keepLastCredentials: 1
    sharedSecretCredentials:
      sharedSecretCredentialsId: "[variables('CREDENTIAL_ID')]"
      secret: "[variables('CREDENTIAL_SECRET')]"
      validToUtc: "[format('{0}-01-01T00:00:00Z', add(date().year, 10))]"

- resourceType: App
  apiVersion: '1.0'
  apiParameters:
    serviceProviderId: "[parameters('SERVICE_PROVIDER_ID')]"
  properties:
    appKey: "[variables('APP_KEY')]"
    name: "[variables('APP_NAME')]"

- resourceType: App/Settings
  apiVersion: '1.0'
  apiParameters:
    serviceProviderId: "[parameters('SERVICE_PROVIDER_ID')]"
    appKey: "[variables('APP_KEY')]"
    sectionName: Logging
  properties:
    settings:
      "LogLevel:Default": Information
      "LogLevel:System": Error


- resourceType: App/Secrets
  apiVersion: '1.0'
  apiParameters:
    serviceProviderId: "[resources('SERVICE_PROVIDER_ID').id]"
    appKey: "[variables('APP_KEY')]"
    sectionName: MyDemoOption
  properties:
    secrets:
      Value1: "test1"
      Value2: "test2"

- resourceType: App/Permissions
  apiVersion: '1.0'
  apiParameters:
    serviceProviderId: "[parameters('SERVICE_PROVIDER_ID')]"
    appKey: "[variables('APP_KEY')]"
    principalId: "[resources('SERVICE_PRINCIPAL').id]"
  properties:
    canReadSettings: true
    canReadSecrets: true


- resourceType: VaultMount
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
  properties:
    mountName: MyDemoVaultMount1
    backendType: kv

- resourceType: VaultMount/Permission
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    mountName: MyDemoVaultMount1
    principalId: "[resources('SERVICE_PRINCIPAL').id]"
  properties:
    permissions:
    - path: "*"
      deny: false
      allowList: true
      allowRead: true
      allowCreate: true
      allowUpdate: true
      allowDelete: true
```

```
- resourceType: VaultMount
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
  properties:
    mountName: MyDemoVaultMount2
    backendType: kv

- resourceType: VaultMount/Permission
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    mountName: MyDemoVaultMount2
    principalId: "[resources('SERVICE_PRINCIPAL').id]"
  properties:
    permissions:
    - path: "*"
      deny: false
      allowList: true
      allowRead: true
      allowCreate: false
      allowUpdate: false
      allowDelete: false

- resourceType: MssqlDatabase
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
  properties:
    databaseName: "[variables('DATABASE_NAME')]"
    options:
      "ALLOW_SNAPSHOT_ISOLATION": ON
      "READ_COMMITTED_SNAPSHOT": ON

- resourceType: MssqlDatabase/CommandExecution
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    databaseName: "[variables('DATABASE_NAME')]"
  properties:
    commands:
    - "[file('1.0.sql')]"
    - "[file('1.1.sql')]"

- resourceType: MssqlDatabase/Permission
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    databaseName: "[variables('DATABASE_NAME')]"
    principalId: "[resources('SERVICE_PRINCIPAL').id]"
  properties:
    timeToLive: '02:00:00'
    permissions:
    - permissions: SELECT, INSERT, UPDATE, DELETE, EXECUTE
      objectType: SCHEMA
      objectName: dbo

- resourceType: ContainerStack
  resourceId: CONTAINER_STACK
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    stackName: "[parameters('GIT_BRANCH')]"
  properties:
    dockerCompose: "[file('docker-compose.yml')]"
    instances: 2
    publicEndpoint:
      serviceName: api
      healthCheck:
        http: "/health"
        interval: '00:00:15'
        timeout: '00:00:10'
    servicePrincipalCredentials:
      id: "[variables('CREDENTIAL_ID')]"
      secret: "[variables('CREDENTIAL_SECRET')]"
  wait:
    until: "[resources('CONTAINER_STACK').isHealthy]"
    interval: '00:00:05'
    timeout: '00:05:00'

- resourceType: ContainerStack/Release
  apiVersion: '1.0'
  apiParameters:
    appKey: "[variables('APP_KEY')]"
    stackName: staging
  condition: "[equals(parameters('GIT_BRANCH'), 'master')]"
  properties:
    deleteExisting: true
```

› Sample: manifest.json (with database)

```
{
   "parameters":[
      {
         "name":"SERVICE_PROVIDER_ID",
         "description":"The identifier of the service provider that owns the app.",
         "type":"string",
         "required":true
      },
      {
         "name":"GIT_BRANCH",
         "description":"The name of the Git branch that triggered the deployment.",
         "type":"string",
         "required":true,
         "defaultValue": "master"
      }
   ],
   "variables":[
      {
         "name":"APP_KEY",
         "value":"DEMO5"
      },
      {
         "name":"APP_NAME",
         "value":"Demo Application 5"
      },
      {
         "name":"DATABASE_NAME",
         "value":"MyTestDatabase_V8"
      },
      {
         "name":"CREDENTIAL_ID",
         "value":"[newGuid()]"
      },
      {
         "name":"CREDENTIAL_SECRET",
         "value":"[newSecret(30)]"
      }
   ],
   "resources":[
      {
         "resourceType":"ServicePrincipal",
         "resourceId":"SERVICE_PRINCIPAL",
         "apiVersion":"1.3",
         "apiParameters":{
            "organizationId":"[parameters('SERVICE_PROVIDER_ID')]"
         },
         "properties":{
            "spn":"[concat(variables('APP_KEY'), '_', parameters('GIT_BRANCH'))]",
            "name":"[concat(variables('APP_KEY'), ' ', parameters('GIT_BRANCH'), ' Service Principal')]",
            "keepLastCredentials": 1,
            "sharedSecretCredentials":{
               "sharedSecretCredentialsId":"[variables('CREDENTIAL_ID')]",
               "secret":"[variables('CREDENTIAL_SECRET')]",
               "validToUtc":"[format('{0}-01-01T00:00:00Z', add(date().year, 10))]"
            }
         }
      },
      {
         "resourceType":"App",
         "apiVersion":"1.0",
         "apiParameters":{
            "serviceProviderId":"[parameters('SERVICE_PROVIDER_ID')]"
         },
         "properties":{
            "appKey":"[variables('APP_KEY')]",
            "name":"[variables('APP_NAME')]"
         }
      },
      {
         "resourceType":"App/Settings",
         "apiVersion":"1.0",
         "apiParameters":{
            "serviceProviderId":"[parameters('SERVICE_PROVIDER_ID')]",
            "appKey":"[variables('APP_KEY')]",
            "sectionName":"Logging"
         },
         "properties":{
            "settings":{
               "LogLevel:Default":"Information",
               "LogLevel:System":"Error"
            }
         }
      },
      {
         "resourceType":"App/Permissions",
         "apiVersion":"1.0",
         "apiParameters":{
            "serviceProviderId":"[parameters('SERVICE_PROVIDER_ID')]",
            "appKey":"[variables('APP_KEY')]",
            "principalId":"[resources('SERVICE_PRINCIPAL').id]"
         },
         "properties":{
            "canReadSettings":true,
            "canReadSecrets":false
         }
      },
      {
         "resourceType":"MssqlDatabase",
         "apiVersion":"1.0",
         "apiParameters":{
            "appKey":"[variables('APP_KEY')]"
         },
         "properties":{
            "databaseName":"[variables('DATABASE_NAME')]",
            "options":{
               "ALLOW_SNAPSHOT_ISOLATION":"ON",
               "READ_COMMITTED_SNAPSHOT":"ON"
            }
         }
      },
      {
         "resourceType":"MssqlDatabase/CommandExecution",
         "apiVersion":"1.0",
         "apiParameters":{
            "appKey":"[variables('APP_KEY')]",
            "databaseName":"[variables('DATABASE_NAME')]"
         },
         "properties":{
            "commands":[
               "[file('1.0.sql')]",
               "[file('1.1.sql')]"
            ]
         }
      },
      {
         "resourceType":"MssqlDatabase/Permission",
         "apiVersion":"1.0",
         "apiParameters":{
            "appKey":"[variables('APP_KEY')]",
            "databaseName":"[variables('DATABASE_NAME')]",
            "principalId":"[resources('SERVICE_PRINCIPAL').id]"
         },
         "properties":{
            "timeToLive":"02:00:00",
            "permissions":[
               {
                  "permissions":"SELECT, INSERT, UPDATE, DELETE, EXECUTE",
                  "objectType":"SCHEMA",
                  "objectName":"dbo"
               }
            ]
         }
      },
      {
         "resourceType":"ContainerStack",
         "resourceId":"CONTAINER_STACK",
         "apiVersion":"1.0",
         "apiParameters":{
            "appKey":"[variables('APP_KEY')]",
            "stackName":"[parameters('GIT_BRANCH')]"
         },
         "properties":{
            "dockerCompose":"[file('docker-compose.yml')]",
            "instances":2,
            "publicEndpoint":{
               "serviceName":"api",
               "healthCheck":{
                  "http":"/health",
                  "interval":"00:00:15",
                  "timeout":"00:00:10"
               }
            },
            "servicePrincipalCredentials":{
               "id":"[variables('CREDENTIAL_ID')]",
               "secret":"[variables('CREDENTIAL_SECRET')]"
            }
         },
         "wait":{
            "until":"[resources('CONTAINER_STACK').isHealthy]",
            "interval":"00:00:05",
            "timeout":"00:05:00"
         }
      },
      {
         "resourceType":"ContainerStack/Release",
         "apiVersion":"1.0",
         "apiParameters":{
            "appKey":"[variables('APP_KEY')]",
            "stackName":"staging"
         },
         "condition":"[equals(parameters('GIT_BRANCH'), 'master')]"
         "properties":{
            "deleteExisting":true
         }
      }
   ]
}
```

Separate Deployment from Release
The sample files release the deployed version into the primary stack as the last step. You can leave that step out if you want to keep the app in a staging stack.

Step 4: Setup Service Provider

You will need to create a new Service Provider with a Deployment Service Principal that will be used later to deploy your team's Microservices under your team's own Service Provider.

Please see Creating Development Team Organisations (Service Providers) for more information.

Step 5: Deploy & Release the App

Deploy the Entire App

Using the Service Layer CLI, run the following command in the script directory:

```
sl deploy manifest.yml -to "mci-api.ci.dimensiondata.cloud" -token "ey..." -param SERVICE_PROVIDER_ID=11112222-3333-4444-5555-666677778888
```

- -to represents the environment you want to deploy your Microservice to
- -token represents the JWT of the Deployment Service Principal created in Step 4
- SERVICE_PROVIDER_ID represents the Id of the Service Provider created in Step 4.

A Linux Docker container image that contains the Service Layer CLI available:

artifactory.devops.itaas-cloud.com:6553/dimensiondata.servicelayer.cli:latest

You will however need to get in touch with the DevOps team from Sydney to gain access to the Docker repository artifactory.devops.itaas-cloud.com:6553.

Configuration Templates
Configuration Templates are idempotent and represent the desired state of the app. Therefore it is safe to execute them multiple times.

You can also deploy the template using the Configuration Template REST API. However, in that case the contents of external files (*docker-compose.yml*, SQL scripts) need to be embedded and the template must be in JSON format.

Instead of providing a token thru the ***-token*** arguement, the CLI also accepts ***-credentialId*** and ***-secret*** arguments for more convenience.

Verify the App

Open a web browser and navigate to the app's public endpoint:

If the app container stack has been released: https://mci-api.ci.dimensiondata.cloud/test1/...

Otherwise: https://mci-api.ci.dimensiondata.cloud/test1-staging/...

Release the Container Stack

If the container stack is not released as part of the deployment (ie.: the application manifest did not define the ContainerStack/Release resource tyoe) , it can be done manually via CLI or API.

```
sl stack release -appKey "DEMO1" -stackName "staging" -to "mci-api.ci.dimensiondata.cloud" -token "ey..." -scaleExisting 0 -deleteExisting
```

Step 6: Clean up

Delete a Specific Container Stack

```
sl stack delete -appKey "DEMO1" -stackName "staging" -from "mci-api.ci.dimensiondata.cloud" -token "ey..."
```

Delete all Container Stacks for an App

```
sl stack delete -appKey "DEMO1" -from "mci-api.ci.dimensiondata.cloud" -token "ey..."
```

No labels