Portal Docs

for Azure Maps

last revised 7/9/2018

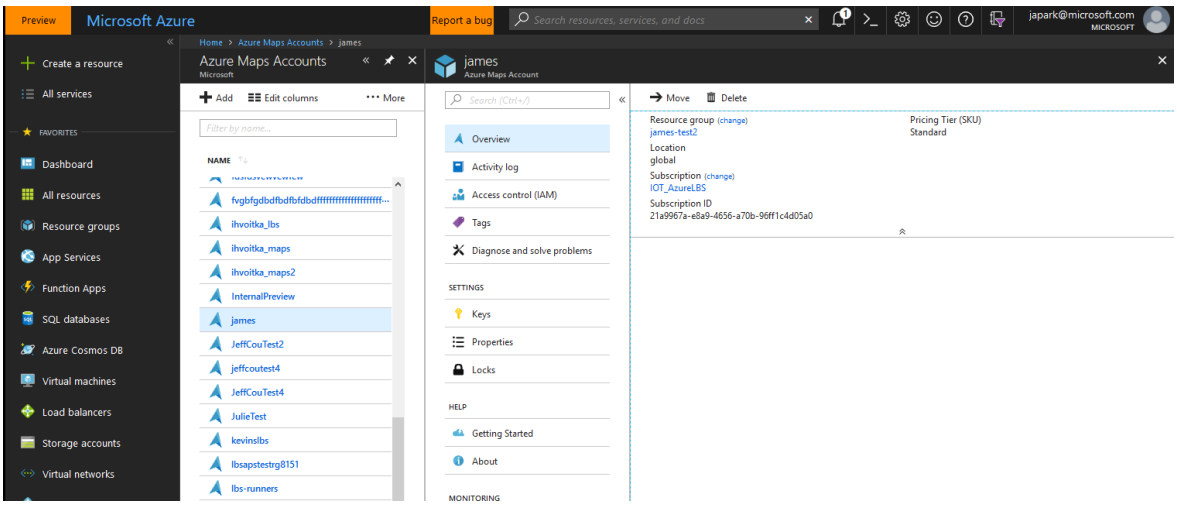
refer to SharePoint

OVERVIEW

Ibiza, known as Azure Portal, is divided into two sections.

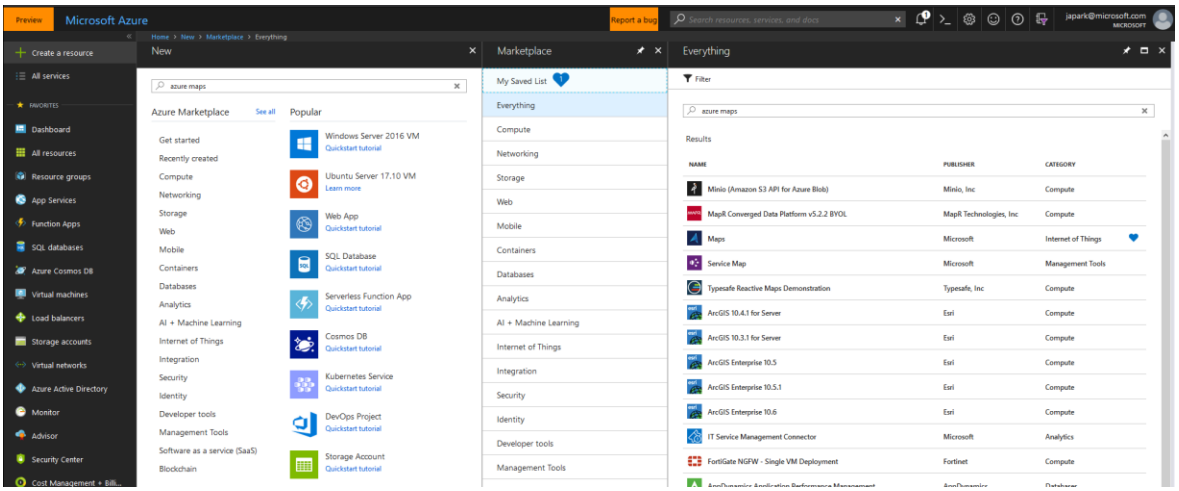
Portal Extension

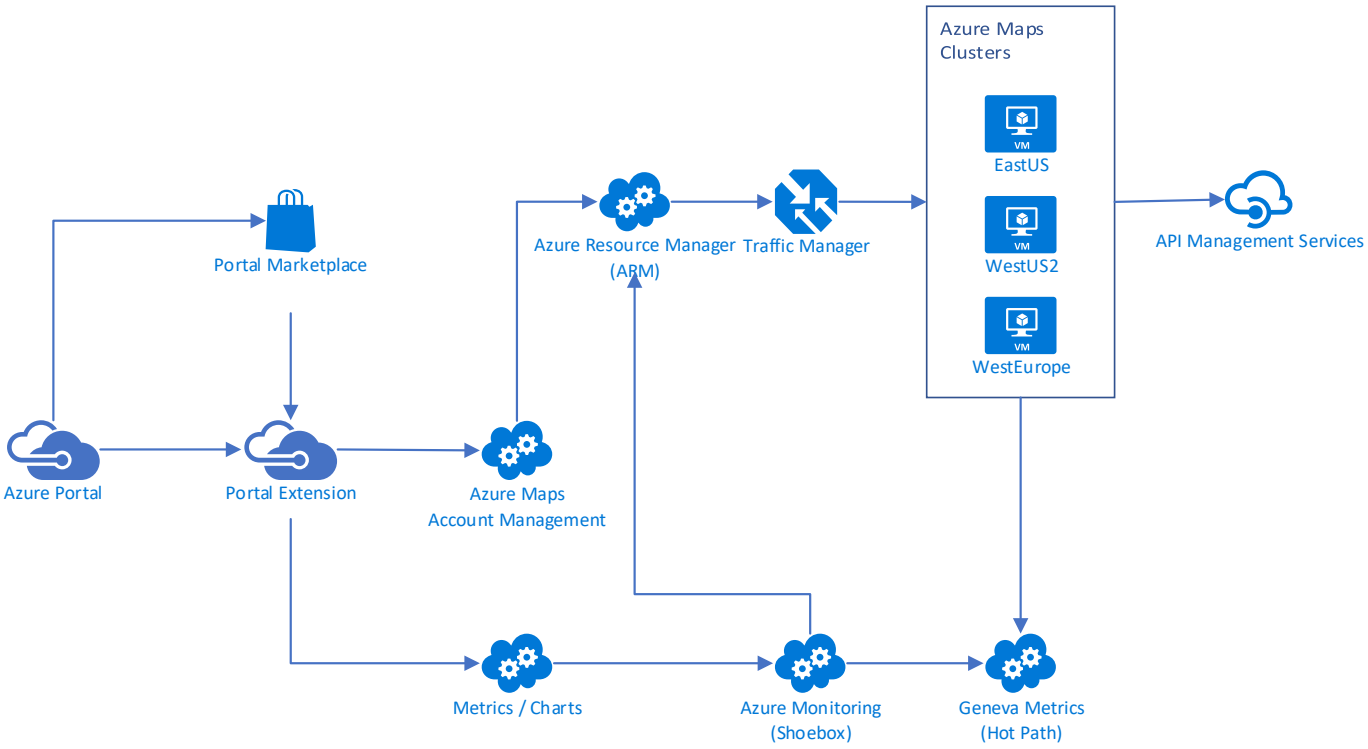
Manage resources (or Azure Maps accounts) experience
Extension is like a service in Azure Portal



Portal Marketplace / Gallery

“Create a resource” experience
Marketplace is like an app store for Azure Portal





Azure Maps exposes account management APIs and a set of statistics for users through Portal.

For any Azure Maps account management APIs, the user interacts with Azure Resource Manager, commonly known as ARM or Sparta. ARM is a service that provisions Azure resources, and is responsible for caching tracked resources, namely Azure Maps accounts. Each service is identified by a Resource Provider namespace in ARM and are defined in the Resource Provider Manifest. The RP Manifest outlines the endpoints for each region and environment to where the call should be passed on to.

Our ARM RP namespace is Microsoft.Maps, and the RP Manifest is available in both Test and Public ARM environments. The manifest is available to view/edit through (prod) Jarvis Actions.

Azure Resource Manager > Resource Provider Management > Get / Put Manifest

Public –management.azure.com
Test – api-dogfood.resources.windows-int.net

We utilize Azure Traffic Manager as our endpoints for manifest to route requests into multiple regions. We have four Traffic Managers, each representing an environment.

CI - c-rp-trafficmgr.trafficmanager.net
Test - t-rp-trafficmgr.trafficmanager.net
Staging - s-rp-trafficmgr.trafficmanager.net
Production - p-rp-trafficmgr.trafficmanager.net

Note: Generally, only Test ARM RP manifest should be allowed to have endpoints changed.
When the endpoint is updated for accounts, it is advisable to invalidate the ARM cache.
This can be done via *Azure Resource Manager > Resource Group Management > Synchronize subscription resources*

When the request is routed through our endpoint, it is passed into our *ResourceProvider > ResourceProviderWebAPI > Controllers* code and proper response is constructed. For account specific operations, Azure Storage, Azure API Management, etc. are also involved from the *Controllers*.

For visualizing users' data, Portal uses MonitorChart to render charts and Azure Monitoring to bring data into the charts.

MonitorCharts - Azure Portal SDK – API reference
Azure Monitoring – https://aka.ms/shoebox

An example usage of MonitorCharts can be found here.
Azure Monitoring takes data from MDM Hot Path and directly presents them in Portal through MonitorCharts. Data exposures are controlled by our GET Operations under metricSpecifications, where the GET Operations defines our MDM accounts, namespaces, metrics, and its dimensions.

Metrics are logged with MDM library (example), and every new Portal metric must include a resource ID dimension.

PORTAL EXTENSION | Documentation

Dev Setup

1. Clone Azure-IoT-LocationBasedServices repository
2. Open *Visual Studio 2017* as administrator
3. Open *Portal* folder and set *LocSvcExtension* as StartUp Project
4. Set as *Debug* and *x64*

Code Structure

`LocSvcExtension`

> `Client`

> `ClientResources.resx`

Where all localizable strings are stored.

`ClientResources.*.resx` files are generated from Simpleloc (see **Localization** section).

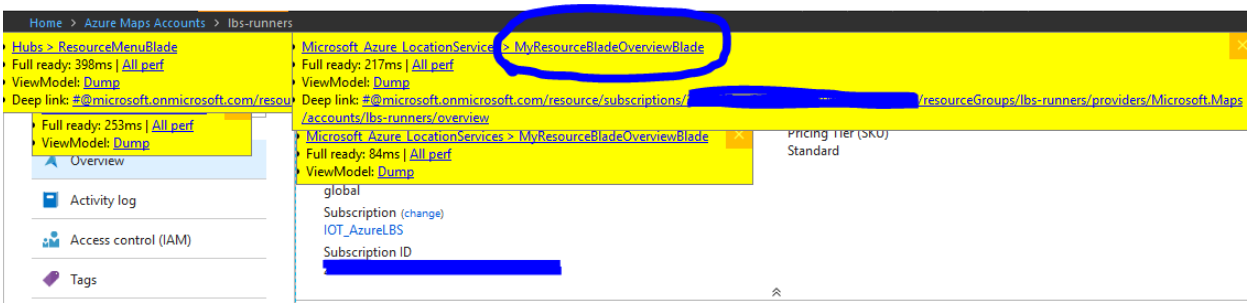
Note: You want to keep all strings, that you wish to be translated, here.

> `MyResource`

Where blades are defined for Portal.

Blades contain custom UI and logic for extensions.

Note: You can find corresponding blade filename by pressing ctrl+alt+d in Portal.



> `Styles`

> `extensions.css` (linked via `extension.pdl`)

Where custom CSS styles are defined for extension blades.

Note: All custom CSS styles must begin with `.ext-*`

> `Templates`

> `MyResourceBladeOverviewBlade.html`

Where html is defined for `MyResourceBladeOverviewBlade.ts`

Note: Other blades define html within blades ts files as they are not complex.

Testing

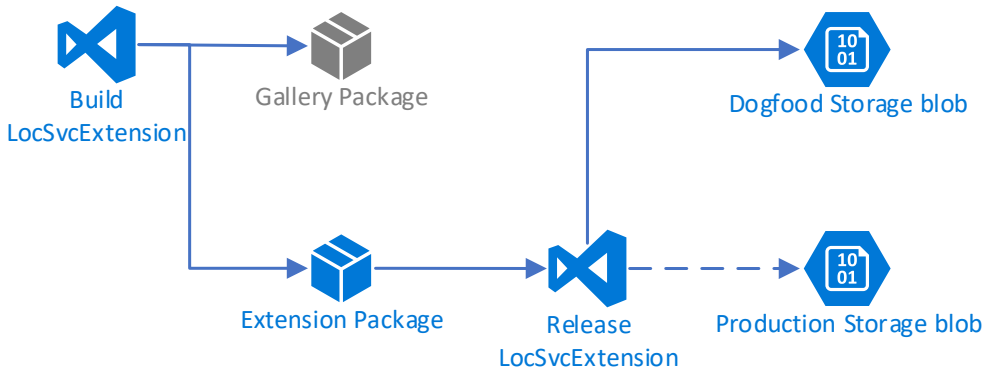
Testing Extension changes can be done locally or through Dogfood Portal.

Local testing in Portal is done by sideloading code into Prod Portal.

1. Build *LocSvcExtension* project
2. Run *LocSvcExtension*
3. Navigate here with a favorite browser
4. Click “Allow” on pop-up related to “Untrusted Extensions!”

Dogfood Portal testing is done by running a build for `IoT.LBS.Portal` or checking in changes to the `master` branch (see **Publishing** section). After a build and release for Dogfood Portal have been completed, visit this page. The changes should be applied within 15 minutes of release. If changes do not occur, make sure that browser cache is cleared.
Note: Also make sure that `?Microsoft_Azure_LocationServices=true` is in the URI for Dogfood Portal. `Microsoft_Azure_LocationServices` is our extension ID.

Publishing



To publish Extension changes live, a build must be started for `IoT.LBS.Portal`.
Generally, the build starts automatically when code is merged into the `master` branch.

> The common guideline is to use:
> `master` branch for Dogfood Portal
> `deployments/[number]` branch for Production Portal

For any changes to take into effect, a unique Portal Extension version must be assigned. This step is done automatically through `Set-Portal-Version.ps1`.

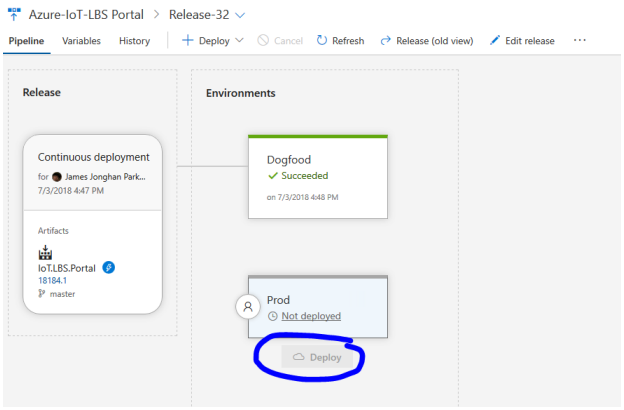
> The version is defined as `x.y.build_number`.
> `x` and `y` values are set in `extension.pdl` and `AssemblyInfo.cs`.
> `build_number` is a uniquely assigned/generated value in each build.
> For major changes, `x` and `y` should be manually modified.

When build completes, it will drop two packages: Gallery (azpkg) and Extension (zip). Release is then created, and the Extension package is uploaded to `azurelsportalextension` Azure Storage (known as PortalFx Extension Hosting Service).

When release completes, it should take up to 15 minutes for the changes to take effect in Dogfood Portal.

Production Portal deployment is done manually, where one must visit the release page and click “Deploy”. Upon manager’s approval, Production Portal changes should take up to 15 minutes.

Note: Portal changes are deployed, by default, to all regions at once. If region-by-region deployments are necessary, contact Khalid Alquinyah and follow the steps here.



PORTAL MARKETPLACE / GALLERY | Documentation

Dev Setup

1. Clone Azure-IoT-LocationBasedServices repository
2. Open *Visual Studio 2017* as administrator
3. Open Portal folder and set *LocSvcExtension* as StartUp Project
4. Set as Debug and x64

5. Successfully build *LocSvcExtension*
6. Navigate to `root/packages/Microsoft.Azure.Gallery.AzureGalleryUtility.[latest_version]/tools`
7. Visit https://github.com/Azure/portaldocs/blob/master/gallery-sdk/templates/gallery-items.md#configuring-the-azure-package-loader-tool and follow the instructions.
   You will be downloading and installing the test certificate to your local machine.
   You will edit the `appSettings` in `AzureGallery.exe.config` within the folder.

Code Structure

`LocSvcExtension`
> `GalleryPackages`
 > `Create`
  > `strings`
   > `resources.resjson`
      Where all localizable strings are stored.
      Folders in strings are generated from *Simpleloc* (see **Localization** section).
      Strings can be referenced by *ms-resource:[string_name]*

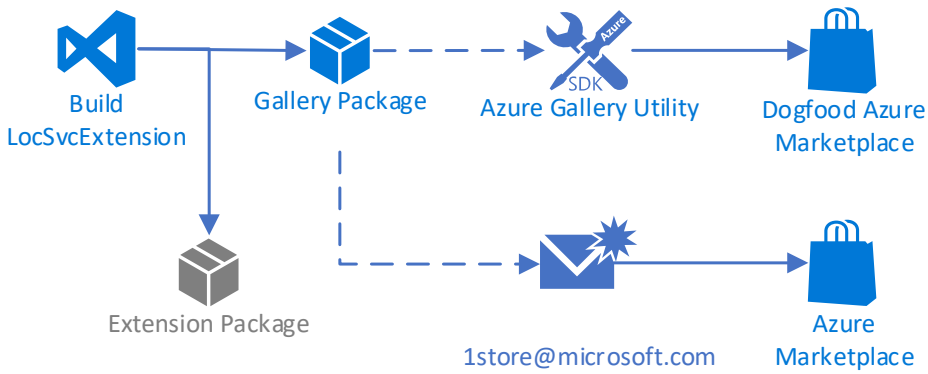
   > `Manifest.json`
      Where gallery package is defined. Refer to docs.

   > `UIDefinition.json`
      Where you define links to other services like Portal Extension and Portal Startboard (Dashboard button).

Testing

Testing Marketplace changes is done through Dogfood Portal (see **Publishing** section).

Publishing



To publish Marketplace/Gallery changes live, a new version number must be assigned.
Azure Marketplace will always render the latest/highest version to clients.

The version can be edited in `Manifest.json` and follows semantic versioning.
The Gallery package can be created by building `LocSvcExtension` locally (via VS 2017).

It can then be found at:

```
> root\src\LocSvcExtension\LocSvcExtension\App_Data\Gallery
```

After completing the **Dev Setup** steps 5 – 7,
use this command to upload the package to Dogfood Portal:

```
> AzureGallery.exe upload -p ..\path\to\package.azpkg
```

It should take at most 30 minutes for changes to take effect.

To publish changes to Production Portal, submit an email to v-thramu@microsoft.com (our default contact) or 1store@microsoft.com with the desired/built azpkg as an attachment.

It should take at most 2 days, but generally completed by end of the day.

Maintenance

Localization

Azure Portal requires that we have string translations for multiple languages. We achieve this compliance by utilizing *SimpleLoc*. *SimpleLoc* monitors any changes to our `ClientResources.resx` and `resources.resjson` in our `master` branch. Localization files are refreshed twice a day; however, actual localization may take up to one week.

Though it's not completely automated, we do have a script that will pull new files and overwrite existing localization files.

1. Create a branch from the `master` branch
2. Run `LocalizeProject.ps1`
3. Submit a PR into the `master` branch after committing changes

SDK Versions

Azure Portal requires that we maintain the latest Portal NuGet packages. The requirement is that we should not have portal packages that are older than 90 days, and Portal will automatically raise a live site incident against us if we do not do so.

Portal packages are unfortunately frequently updated with breaking changes. Breaking changes are noted portaldocs; however, it often runs days behind the latest package versions. Sometimes the breaking changes are not noted as well…

To update to the latest Portal packages,

1. Create a branch from the `master` branch
2. Make sure that breaking changes are addressed
3. Open NuGet Package Manager for *LocSvcExtension* and *LocSvcExtension.DataModels*
4. Click on "Updates" tab, and filter by *Microsoft.Portal*
5. Update Portal packages
6. Make sure Portal is working as intended
7. Submit a PR into the `master` branch after committing changes
8. Portal package updates are applied only when the changes are published to Dogfood and Prod Portal (see **Publishing** section in Portal Extension)

Misc

Portal Reference
https://aka.ms/portaldocs

Portal Extension API Reference
https://df.onecloud.azure-test.net/#blade/SamplesExtension/SDKMenuBlade/apiReference