Automating Image Management and Dissemination using Python

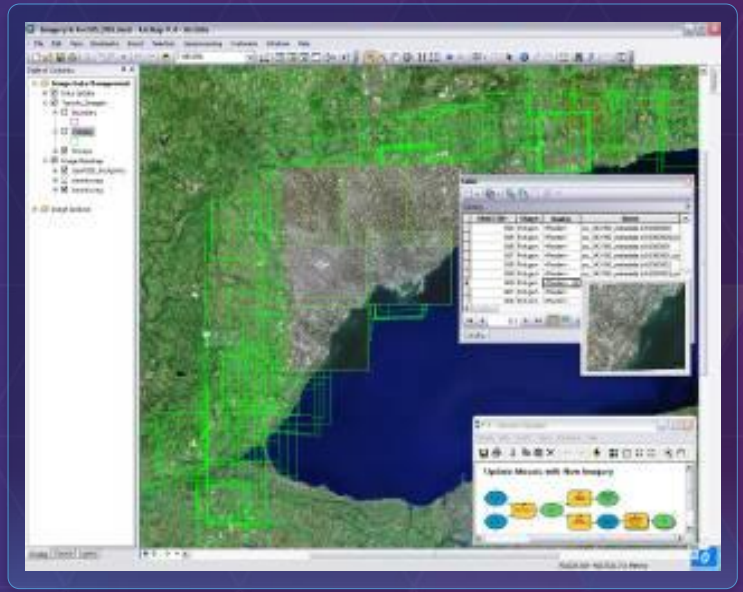
Jamie Drisdelle, Zoey Zou

Overview

- Introduction to mosaic dataset and raster product
- Automate mosaic dataset authoring workflow with python
  - To discover metadata in raster dataset
  - To create and add data to mosaic dataset
  - To configure mosaic dataset
- Introduction to image service
- Automate publishing/updating of image service with python
- Making image service REST request in python

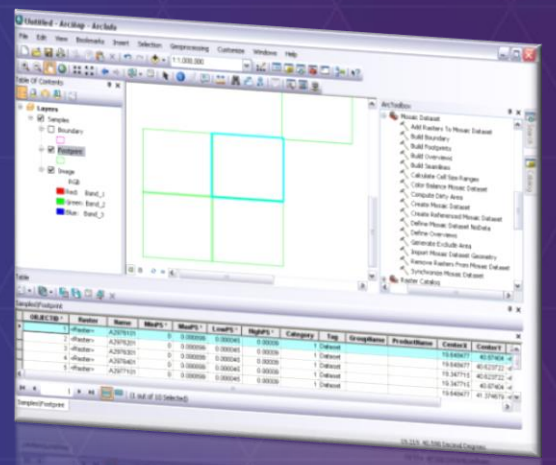
Mosaic Dataset

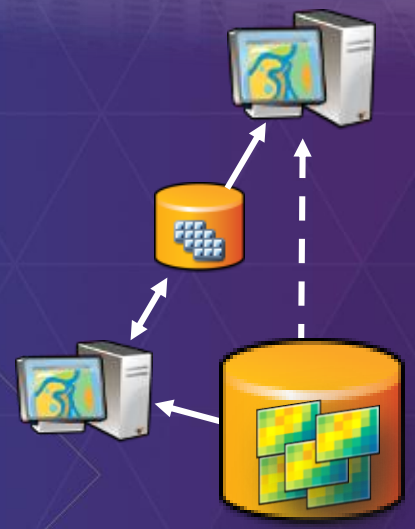
- A geodatabase data model used to catalog and process your collections of imagery
  - Stored as a table and viewed as a table or image
- Indirect pixel management
  - Images can remain in their native format on disk or be loaded into the geodatabase
- Unlimited size*
- Provides dynamic mosaicking and on-the-fly processing
- License requirement – Standard or Advanced



Building a mosaic dataset

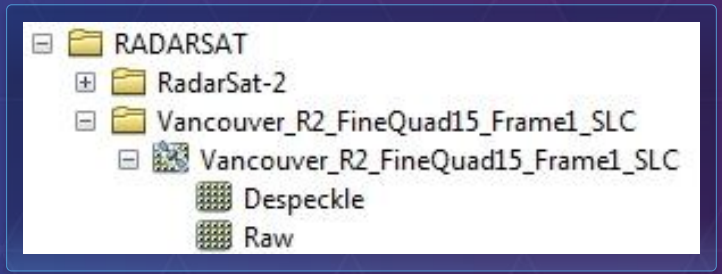
- Store in a geodatabase
  - Build with geoprocessing tools
  - Automation with models or Python
- Simple workflow
  1. Create mosaic dataset
  2. Add imagery (raster type)
  3. Optionally, edit properties and functions
- Can interactively edit and view in ArcMap
  - All layers are displayed
  - Edit and add fields in table window





Raster products

- Simplifies using sensor data
  - Quick and easy visualization of common band combinations
  - Simple drag-n-drop, less clicking
- Key metadata
  - Sensor name
  - Acquisition date
  - Wavelength
- Function templates
  - Multispectral, Pansharpen
- Temporary function raster dataset



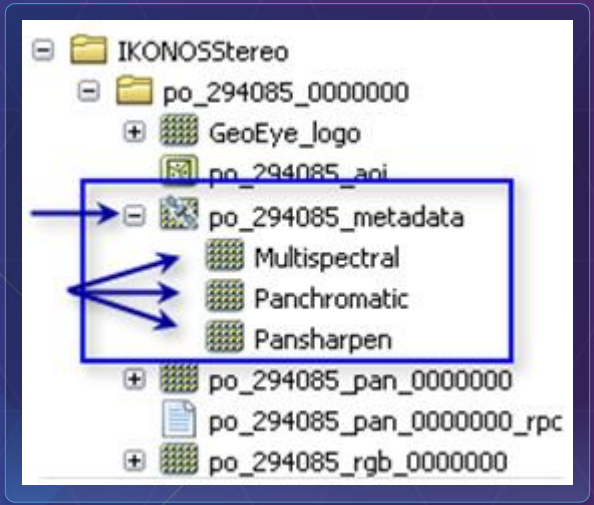
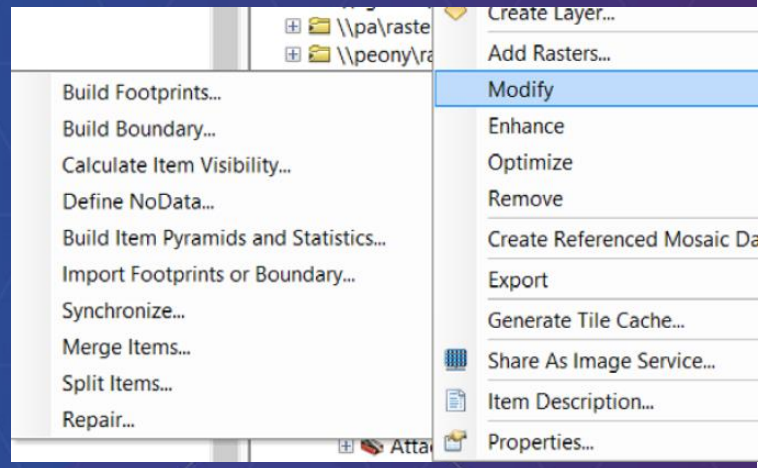
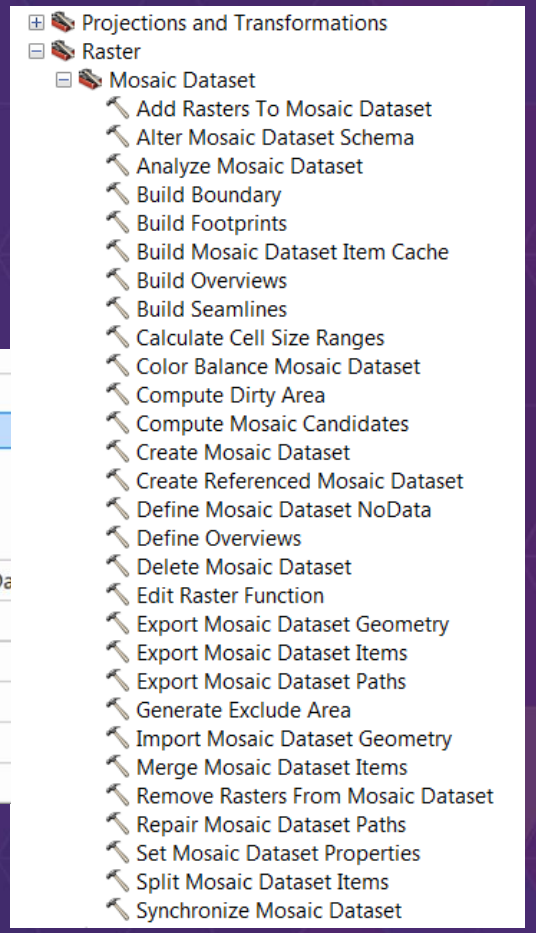


Image collection management with Mosaic Dataset

- Geoprocessing tools
  - Creation
    - Create Mosaic Dataset
    - Add Rasters
    - Remove Rasters
  - Enhancement
    - Define Mosaic Dataset Nodata
    - Build Footprints
    - Build Seamlines
    - Color Balancing
  - Optimize
    - Build Overview
- All tools are accessible through arcpy





Automate Mosaic Dataset Creation with Python

Create Mosaic Dataset Basic

Discover the type of image data, use proper raster type

- Find raster data in your workspace

```
arcpy.env.workspace = workspace
rasterds = arcpy.ListRasters()
for raster in rasterds:
    yield os.path.join(workspace, raster)
```

- Check property to find data type

```
#get the sensorName property from the raster dataset
sensorNameResult = arcpy.GetRasterProperties_management(
    raster, "SENSORNAME")
```

- Add data to mosaic dataset with the correct **type**

```
# create mosaic dataset
arcpy.env.overwriteOutput = 1
arcpy.CreateMosaicDataset_management(gdbName, mdName, "54004")

# load data for this raster type
arcpy.AddRastersToMosaicDataset_management(
    os.path.join(gdbName, mdName), rasterType, indir)
```

Create Derived Mosaic Dataset

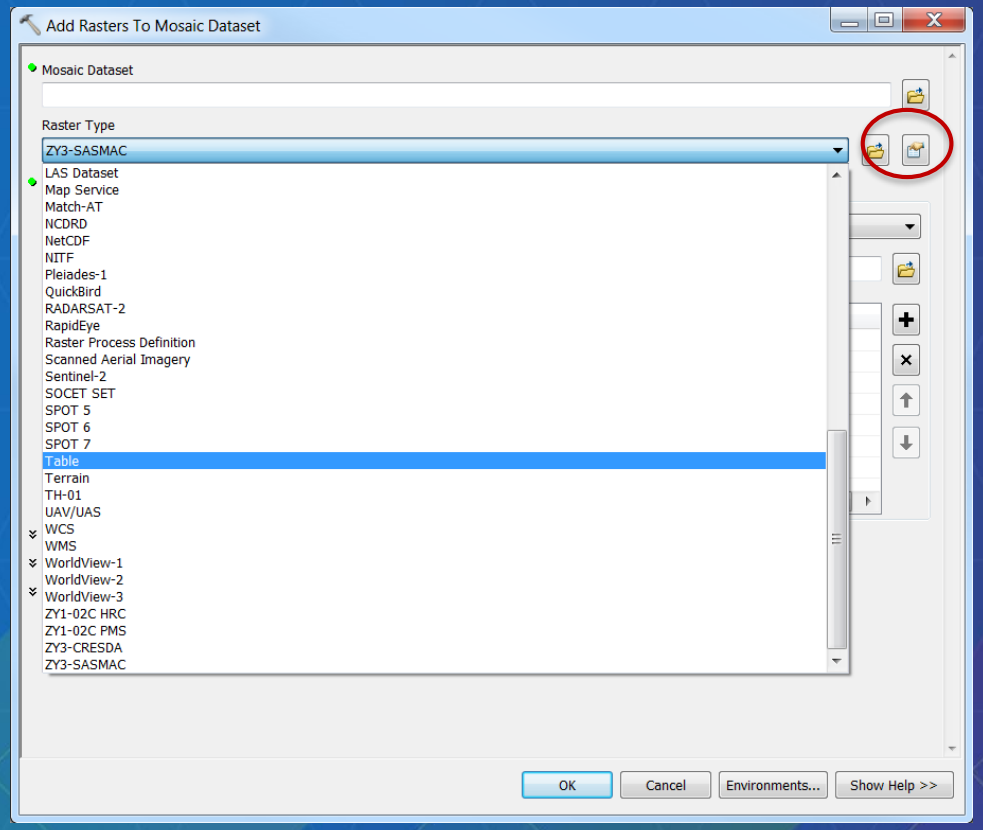
- Create derived mosaic dataset
  - Create derived mosaic dataset for specific project
  - Add data from existing mosaic dataset to a new mosaic dataset
  - Use Table raster type

```
# Add rasters using the Table raster type file
# to create derived mosaic dataset
mdpath = os.path.join(
    arcpy.env.workspace, "FGDB.gdb/Geoeye1nIKONOS")
inputgeoeye = r"e:\MDGDB.gdb\sensorType2"
arcpy.AddRastersToMosaicDataset_management(
    mdpath, "Table", inputgeoeye)
```

Create Derived Mosaic Dataset

Modify raster type settings

- Raster type settings are saved as art.xml file




```
<?xml version="1.0"?>
- <RasterType xsi:type="typens:RasterType" xmlns:xsi="http://www.w3.org/
  + <Names xsi:type="typens:ArrayOfString">
  - <Values xsi:type="typens:ArrayOfAnyType">
    + <AnyType xsi:type="typens:TableBuilder">
    + <AnyType xsi:type="typens:ArrayOfItemTemplate" xmlns:typens="htt
      <AnyType xsi:type="xs:string">Geoeye_table2</AnyType>
      <AnyType xsi:type="typens:ArrayOfString"/>
      <AnyType xsi:type="xs:int">1</AnyType>
      <AnyType xsi:type="xs:string">Supports all tables</AnyType>
      <AnyType xsi:type="xs:int">176</AnyType>
      <AnyType xsi:type="xs:string">"Tag"='Pansharpened'</AnyType>
      <AnyType xsi:type="xs:boolean">false</AnyType>
      <AnyType xsi:type="xs:boolean">false</AnyType>
      <AnyType xsi:type="xs:boolean">true</AnyType>
      <AnyType xsi:type="xs:boolean">true</AnyType>
      <AnyType xsi:type="xs:boolean">false</AnyType>
      <AnyType xsi:type="xs:boolean">false</AnyType>
      <AnyType xsi:type="xs:boolean">true</AnyType>
    - <AnyType xsi:type="typens:UID">
        <UID xsi:type="xs:string">{8F2800F4-5842-47DF-AD1D-2077A
      </AnyType>
      <AnyType xsi:type="typens:ArrayOfArgument"/>
    - <AnyType xsi:type="typens:RasterTypeName">
        <Name>Table</Name>
```

Create Derived Mosaic Dataset

Modify raster type settings example

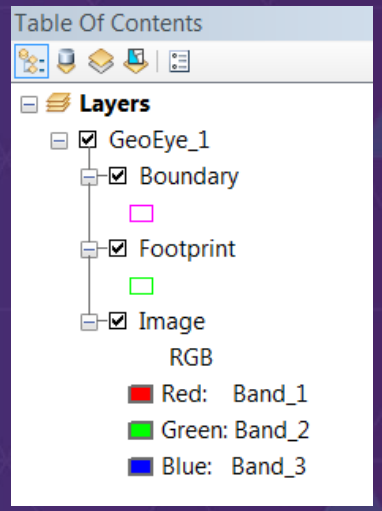
```
#Read raster type setting from xml file
#Update table raster type filter setting
tfiltertxt = "\"Tag\"=\'Pansharpened\'"
from xml.dom import minidom
dom = minidom.parse(rastypepath)
vals = dom.getElementsByTagName('Values')
for val in vals:
    if val.parentNode.tagName == 'RasterType':
        # modify the filter for table raster type
        if val.childNodes[7].firstchild == None:
            val.childNodes[7].appendChild(
                dom.createTextNode(tfiltertxt))
        else:
            val.childNodes[7].firstChild.replaceWholeText(tfiltertxt)

xml_filew = open(rastypepath, "w")
xml_filew.write(dom.toxml())
xml_filew.close()
```

Modify Mosaic Dataset

Modify mosaic dataset in catalog view

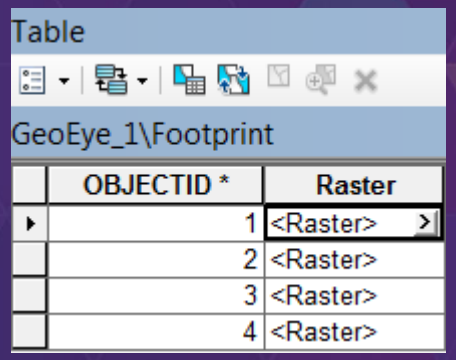
- **Add/Join/Query new attributes to mosaic dataset tables**

```
mdpath = os.path.join(
    arcpy.env.workspace, "FGDB.gdb/Geoeye1nIKONOS")
lutpath = os.path.join(
    arcpy.env.workspace, "FGDB.gdb/lookuptable")
arcpy.AddField_management(
    mdpath, "W_BLUE_MIN", "DOUBLE")
arcpy.JoinField_management(
    mdpath, "Field1", lutpath, "Field2", "ProductName")
```



- **Access mosaic dataset raster item in raster field**

```
rasfields = ["OBJECTID", "Raster"]
with arcpy.da.SearchCursor(mdpath, rasfields) as rcursor:
    for row in rcursor:
        #Create Raster object directly from cursor
        bluemin = arcpy.GetRasterProperties_management(
            row[1], "WAVELENGTH", "BLUE")
```



OBJECTID *	Raster
1	<Raster>
2	<Raster>
3	<Raster>
4	<Raster>

Optimize Mosaic Dataset

More configuration

- Define Nodata & Build Pyramids & Calculate Stats
- Build Seamlines & Apply Color Correction
- Build Overviews

```
#Define nodata value to take out the black border of the image
nodataMode = "COMPOSITE_NODATA"
arcpy.DefineMosaicDatasetNoData_management(
    mdpath, "4", "ALL_BANDS 0", "", "", nodataMode)

#Build Pyramids and Statistics & Color Correction
arcpy.BuildPyramidsandStatistics_management(
    mdpath, "NONE", "NONE", "CALCULATE_STATISTICS", "NONE", "", "", "100", "100")
arcpy.ColorBalanceMosaicDataset_management(mdpath, "DODGING", "COLOR_GRID")

#Set mosaic dataset property "mosaic operator"
mosaicops = "BLEND"
arcpy.SetMosaicDatasetProperties_management(
    mdpath, mosaic_operator=mosaicops)

#Build Overviews
ovrfolder = os.path.join(arcpy.env.workspace, "GeoeyeIKONOSOvr")
arcpy.DefineOverviews_management(mdpath, ovrfolder)
arcpy.BuildOverviews_management(mdpath)
```

Prepare Mosaic Dataset for Live Update

- Image Service places share lock on mosaic dataset
- Use Enterprise Geodatabase Mosaic Dataset
- No change of schema or create/delete table allowed
  - Prepare boundary for future data
  - Prepare fields and tables with Alter Mosaic Dataset Schema tool
    - Fields for different raster types
    - Tables for overviews, etc.
  - Cannot change mosaic dataset properties while serving
    - Number of bands
    - Pixel type
    - Cell size etc.

Demo

What can you do with an image service?

- Use it as an image (visual analysis)
- Use it as raster data (pixel analysis)
- Access it as a catalog (mosaic dataset)

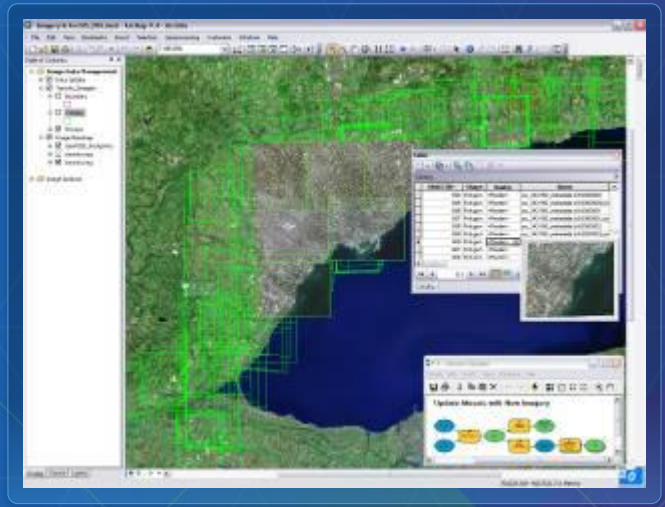
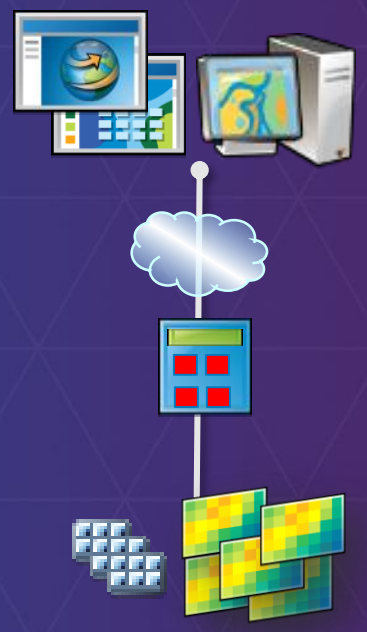


Image service source data

- Data sources
  - Raster datasets
  - Mosaic datasets
    - Requires ArcGIS Server Image Extension
  - Raster or mosaic layers
    - To control rendering
    - Preset some layer properties
    - Predefined query

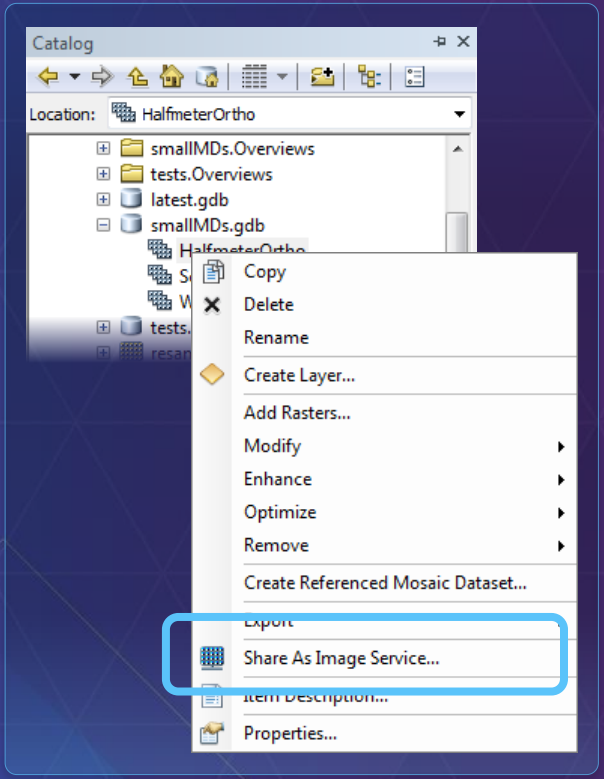


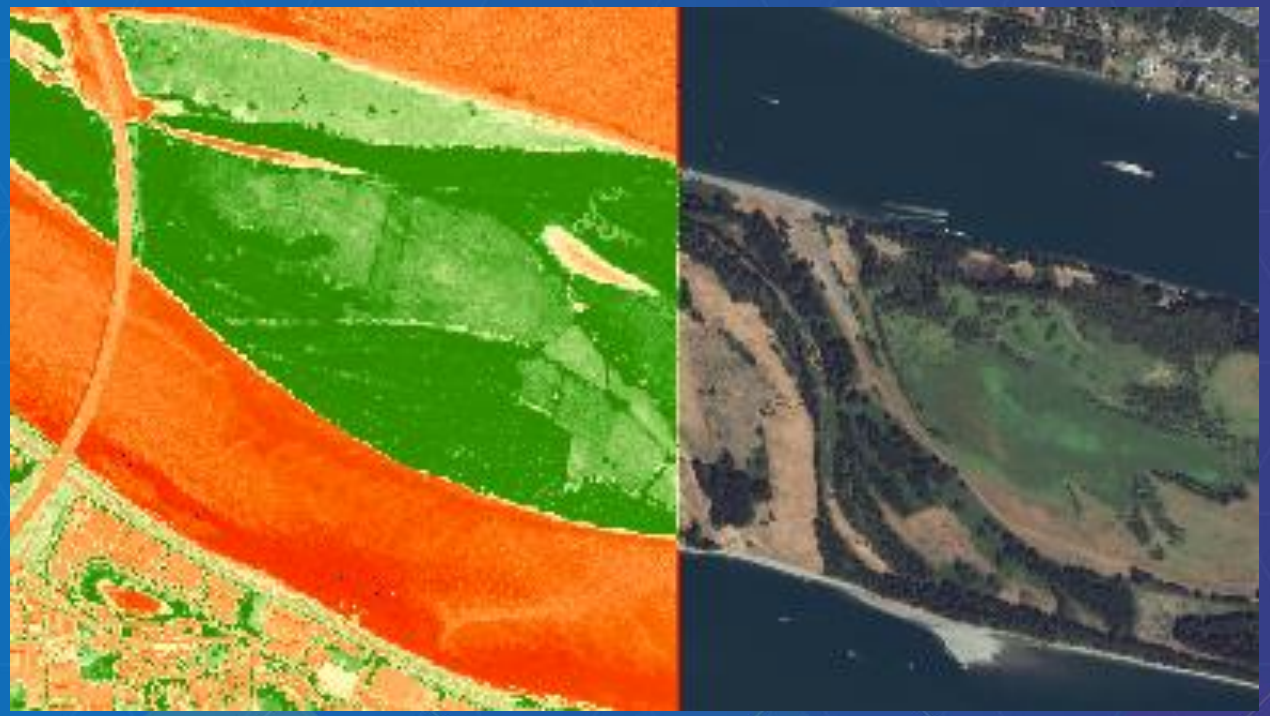
How can you access an image service?

- ArcGIS Desktop
- ArcGIS Pro
- ArcGIS Explorer
- Web APIs (Silverlight, Flex, JavaScript)
- ArcGIS.com
- REST, SOAP
- WMS, WCS, KML
- 3rd Party Applications

Publishing an image service

- New publishing workflow
- Register databases
- Share from data source
- Requires service definition (.sd)





Publish and Update Image Service with Python

Imagery Sharing

Create image service definition draft

- **Create publisher server connection file**

```
conType = "PUBLISH_GIS_SERVICES"
folderPath = os.path.join(os.getcwd(), "output")
fileName = serverName + "_publisher.ags"
serverURL = "http://"+serverName+":6080/arcgis"
serverType = "ARCGIS_SERVER"

arcpy.mapping.CreateGISServerConnectionFile(
    conType, folderPath, fileName, serverURL, serverType,
    username=userName, password=passWord)
```

- **Create image service definition draft**

```
sddraftPath = os.path.join(
    folderPath, serviceName+".sddraft")
arcpy.CreateImageSDDraft(
    mdPath, sddraftPath, serviceName, "ARCGIS_SERVER",
    copy_data_to_server=False)
```

Imagery Sharing

Configure service setting

- A sample *.sddraft file


```
  <ClientHostName>SKYE</ClientHostName>
  <OnServerName/>
- <Configurations xsi:type="typens:ArrayOfSVCConfiguration">
  - <SVCConfiguration xsi:type="typens:SVCConfiguration">
      <ID>764C67C2-E070-42C6-A03A-9EC0DD55C592</ID>
      <Name>Vancouver</Name>
      <TypeName>ImageServer</TypeName>
      <ResourceID>{46E4624F-FBBE-436C-B584-E8302FA56C79}</ResourceID>
      <ServiceFolder/>
      <DataFolder/>
    - <Definition xsi:type="typens:SVCConfigurationDefinition">
        <Description/>
      - <ConfigurationProperties xsi:type="typens:PropertySet">
        - <PropertyArray xsi:type="typens:ArrayOfPropertySetProperty">
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>path</Key>
              <Value xsi:type="xs:string">e:\demo\2013DevSummit\MD\demo2\FGDB.gdb\Geoeye1nIKONOS</Value>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>description</Key>
              <Value xsi:type="xs:string"/>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>supportedImageReturnTypes</Key>
              <Value xsi:type="xs:string">URL</Value>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>cacheDir</Key>
              <Value xsi:type="xs:string"/>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>useLocalCacheDir</Key>
              <Value xsi:type="xs:string">true</Value>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>isCached</Key>
```

Imagery Sharing

Edit service setting

- **Add a raster function template to the service definition**

```
functemp = "\\\\server\\datastore\\function_template.rft.xml,None"
xml = sddraftPath
dom = DOM.parse(xml)
# Add a NDVI raster function tempalate
properties = dom.getElementsByTagName('PropertySetProperty')
for prop in properties:
    keynodes = prop.getElementsByTagName("Key")
    for keynode in keynodes:
        # Check the key-value pair which stores the raster function setting
        if keynode.firstChild.nodeValue == "rasterFunctions":
            valnodes = prop.getElementsByTagName("Value")
            for valnode in valnodes:
                if valnode.firstChild == None:
                    valnode.appendChild(dom.createTextNode(functemp))
                else:
                    valnode.firstChild.replaceWholeText(functemp)
```

Imagery Sharing

Analyze service definition draft

```
analysis = arcpy.mapping.AnalyzeForSD(sddraftPath)

for key in ('messages', 'warnings', 'errors'):
    print "----" + key.upper() + "---"
    vars = analysis[key]
    for ((message, code), data) in vars.iteritems():
        print "    ", message, " (CODE %i)" % code
```

Imagery Sharing

Stage and publish

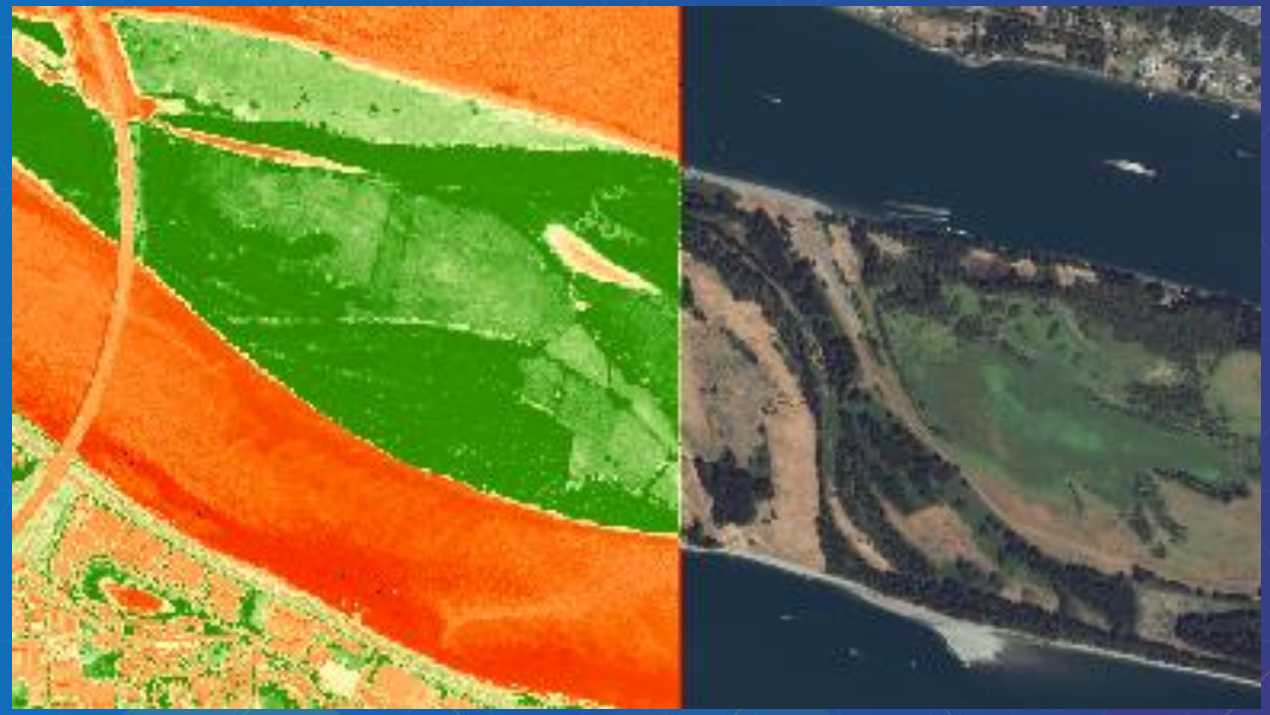
- **Stage *.sddraft file to service definition *.sd file**

```
sdPath = sddraftPath.replace(".sddraft", ".sd")
arcpy.StageService_server(sddraftPath, sdPath)
```

- **Publish service definition file to ArcGIS Server**

```
arcpy.UploadServiceDefinition_server(sdPath, connectionfile)
```

Demo



Use Image Service REST API with Python

Image Service REST API

What is in REST API

- Get service information
- Query item
- Export Image
  - Define geometry
  - Define mosaic rule
    - LockRaster to export from specific item
  - Support compression
  - Request different rendering rules
  - Export format
    - Only TIFF format keep spatial reference information

REST API reference:
https://servername:6443/arcgis/sdk/rest/index.html

Image Service REST API

Make REST request – the standard way

- **Construction request in JSON**

```
# Read json data from the file
data = open(json_file).read()
json_dict = json.loads(data)
json_data = json.dumps(json_dict)
serviceName = json_dict["serviceName"]

# Construct REST request content
content = "f=pjson&token="+token+"&service="+json_data
post_data = content

headers = {}
headers["Content-Type"] = "application/x-www-form-urlencoded"
```

- **Submit request and get response with urllib2**

```
# Construct create service REST url
adminURL = "http://"+serverName+":6080/arcgis/admin/services/createService/"+folderName

# Publish image service to the server
req = urllib2.Request(adminURL, post_data, headers)
response_stream = urllib2.urlopen(req)
response = response_stream.read()

# Check response string
if response.find("success") > 0:
    arcpy.AddMessage("Successfully published service.")
```

Demo

Questions?