WORKSHOP HARRIS GEOSPATIAL SOLUTIONS ITALIA

Stefano Gagliano

September, 13th, 2018 - INAF

IDL From the Desktop to the Enterprise and Solutions providers

Presentation and Discussion

- IDL Quick Overview
- GSF: Geospatial Services Framework: our framework for Cloud services
- IDL Task System: creating tasks to be consumed by services engines in GSF
- ENVI Modeler: the visual tool to create your own workflows, from the Desktop to the Enterprise
- The Evolution of Harris Geospatial
- Scalability & Bigdata Processing
- Q&A

HARRIS®

IDL

DISCOVER WHAT'S IN YOUR DATA

> Language for Analysis, intuitive and powerful

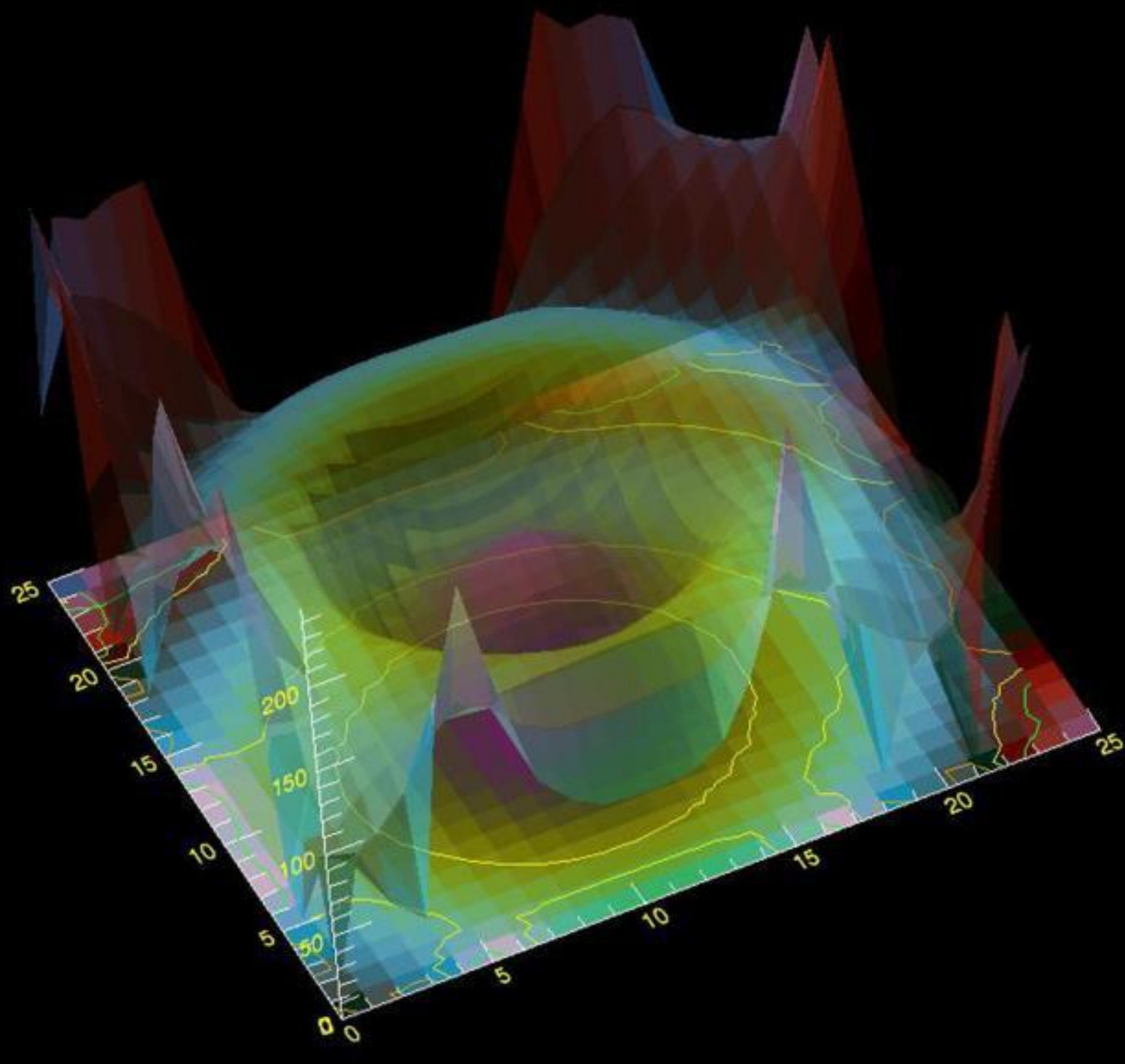
> Interactive Graphics System

> Development Environment

> *IDL-Python Bridge*

> Asynchronous Job Classes

> Output File Formats



IDL

DISCOVER WHAT'S IN
YOUR DATA

> **Language for Analysis, intuitive and powerful**

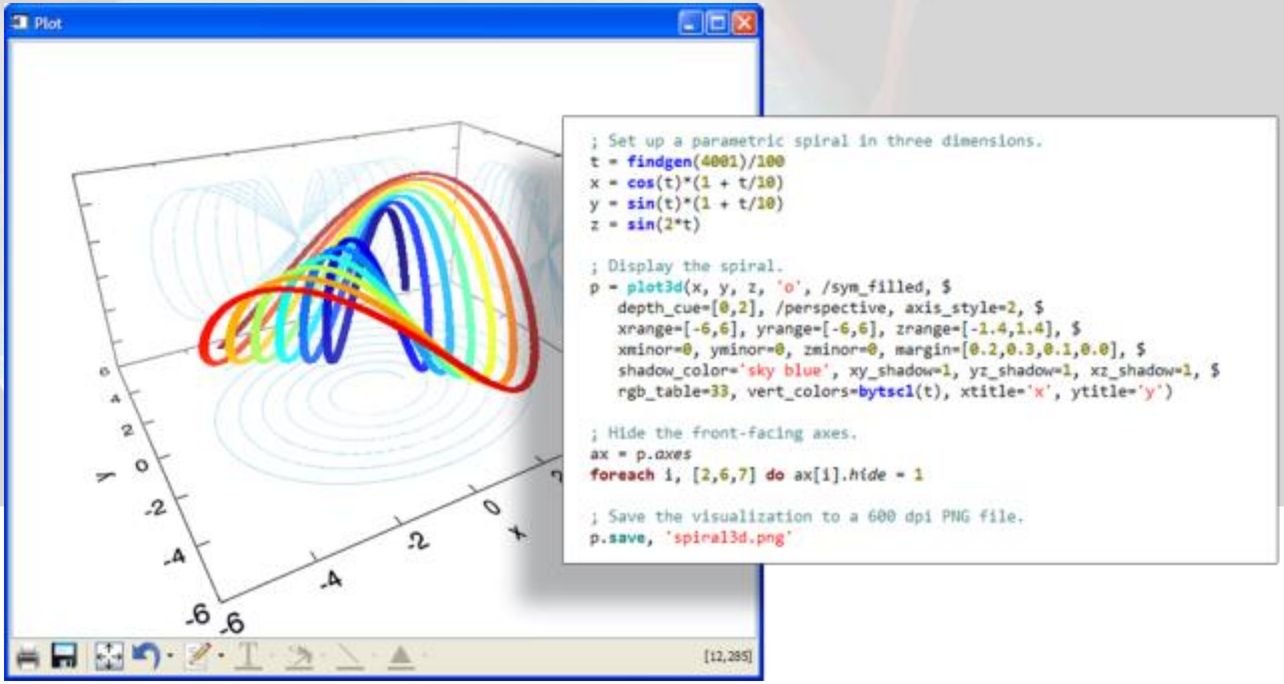
> Interactive Graphics System

> Development Environment

> *IDL-Python Bridge*

> Asynchronous Job Classes

> Output File Formats



Language for analysis, intuitive and powerful

IDL

DISCOVER WHAT'S IN YOUR DATA

> Language for Analysis, intuitive and powerful

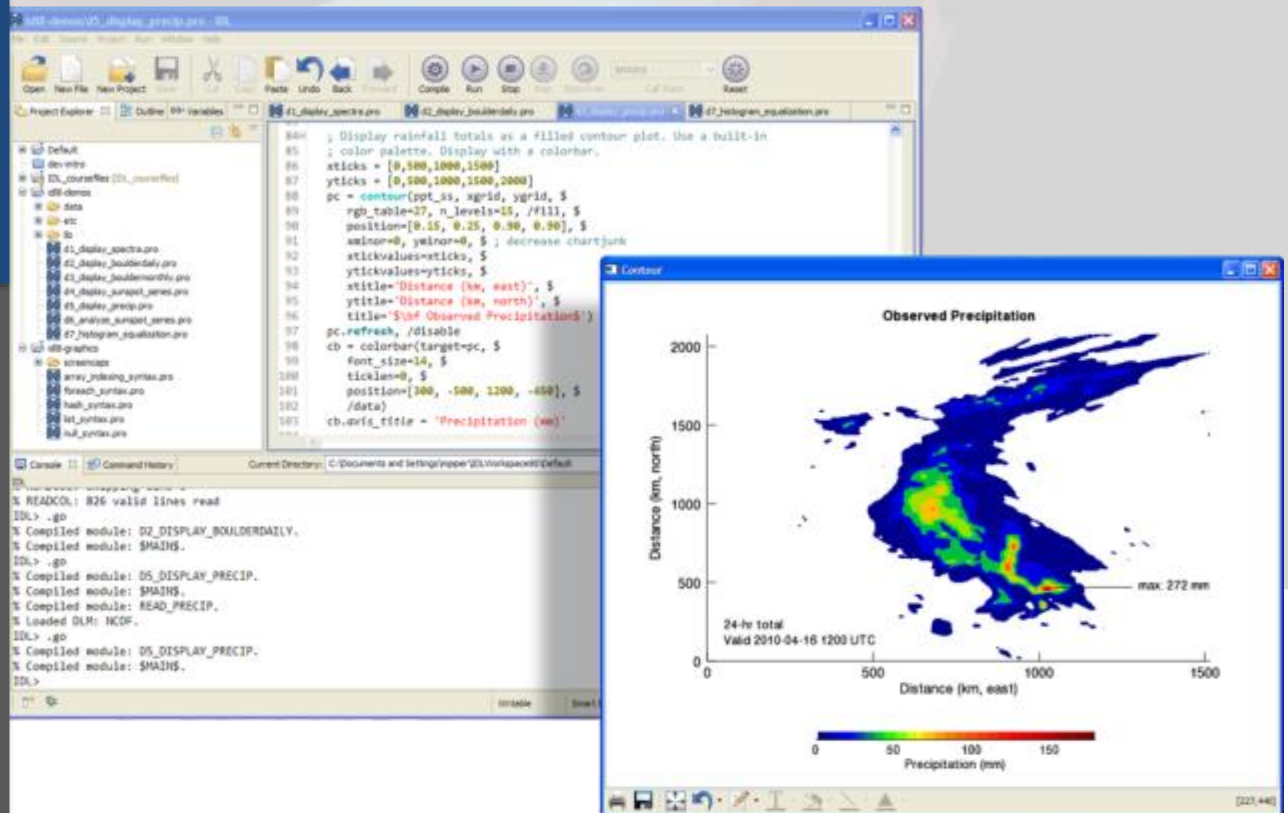
> **Interactive Graphics System**

> Development Environment

> *IDL-Python Bridge*

> Asynchronous Job Classes

> Output File Formats

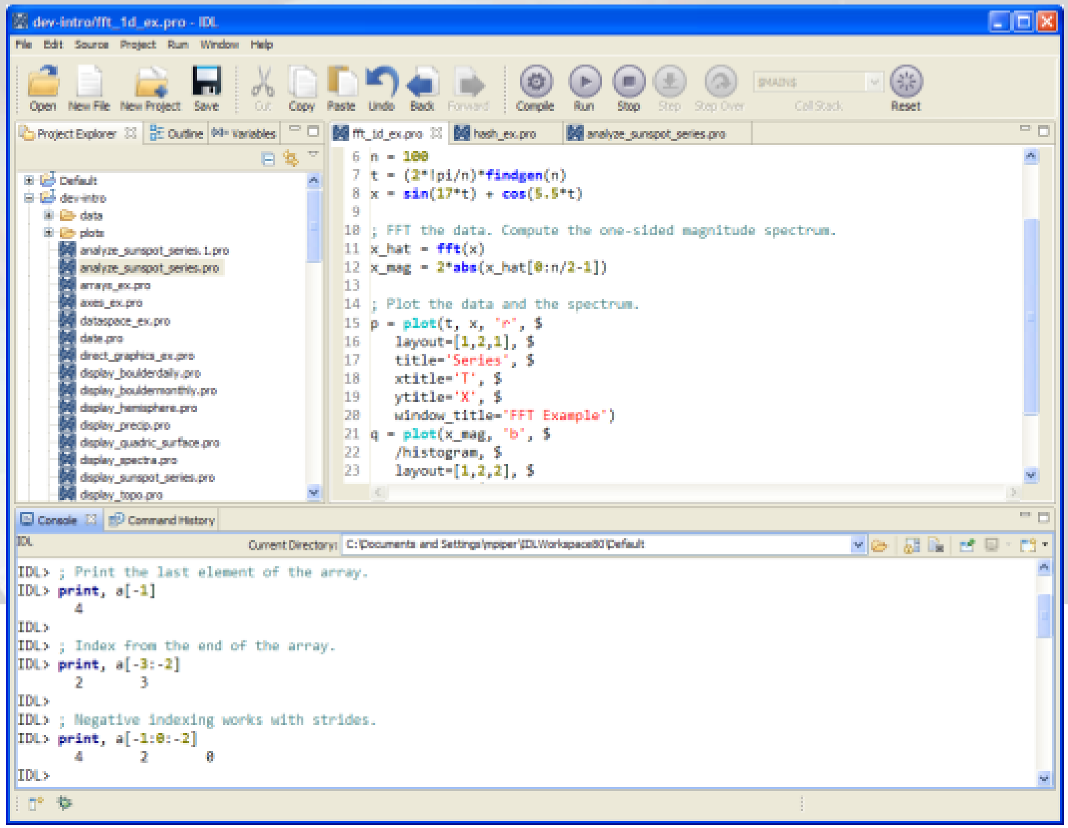


Interactive Graphics System

IDL

DISCOVER WHAT'S IN YOUR DATA

- > Language for Analysis, intuitive and powerful
- > Interactive Graphics System
- > **Development Environment**
- > *IDL-Python Bridge*
- > Asynchronous Job Classes
- > Output File Formats



Development Environment

HARRIS

IDL

DISCOVER WHAT'S IN YOUR DATA

> Language for Analysis, Rules, and Conventions

> Interactive Graphics System

> Development Environment

> **IDL-Python Bridge**

> Asynchronous Job Classes

> Output File Formats

Python Code Snippet

```python
from idlpy import IDL
import numpy as np
import os

#start ENVI
e = IDL.envi(HEADLESS = 1)

# get task definition from IDL
task = IDL.ENVITask("BuildMosaicRaster")
task.INPUT_RASTERS = rasters
task.RESAMPLING = 'Nearest Neighbor'
task.FEATHERING_METHOD = 'edge'
task.OUTPUT_RASTER_URI = e.GetTemporaryFilename()
task.execute()
```

IDL Code Example

```
; Define some IDL variables
labels = ['Baltam', 'Python', 'IDL', 'Other']
sizes = [20, 30, 40, 10]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
explode = [0, 0, 0.1, 0] ; "explode" the 3rd slice


; Import some Python modules
pyplot = Python.Import('matplotlib.pyplot')

; Call methods on the Python modules
pie = pyplot.pie(sizes, explode=explode, $
  labels=labels, colors=colors, $
  autopct='%1.1f%%', /shadow, startangle=90)
void = pyplot.axis('equal')
void = pyplot.savefig("myplot.png", dpi = 96)
void = pyplot.show()
```

- **Bi-directional bridge lets you easily call Python from IDL or run IDL from Python across platforms**

IDL - Python Bridge

HARRIS

IDL

DISCOVER WHAT'S IN YOUR DATA

Asynchronous Job Classes

The IDLAsync classes allow you to specify units of work to execute asynchronously outside the main IDL session. For a m

- IDLAsyncBridgeJob: Represents a unit of work to be done at some point in the future inside an IDL_IDLBridge.
- IDLAsyncBridgeTaskJob: Allows the user to specify a single IDLTask that will executed inside an IDL_IDLBridge.
- IDLAsyncJob: Represents a unit of work to be done at some point in the future.
- IDLAsyncJoin: Observes one or more IDLAsyncJob objects to know when they are done.
- IDLAsyncQueue: Manages a collection of IDLAsyncJob objects that are to be executed at some point in the future.
- IDLAsyncSpawnJob: Represents a unit of work to be done at some point in the future by spawning an external pro
- IDLAsyncSpawnTaskJob: Allows the user to specify a single IDLTask that will executed by the TaskEngine.
- IDLTaskJob: Provides an interface for any job that wants to run an IDLTask.

Asynchronous Job Classes

IDL

DISCOVER WHAT'S IN YOUR DATA

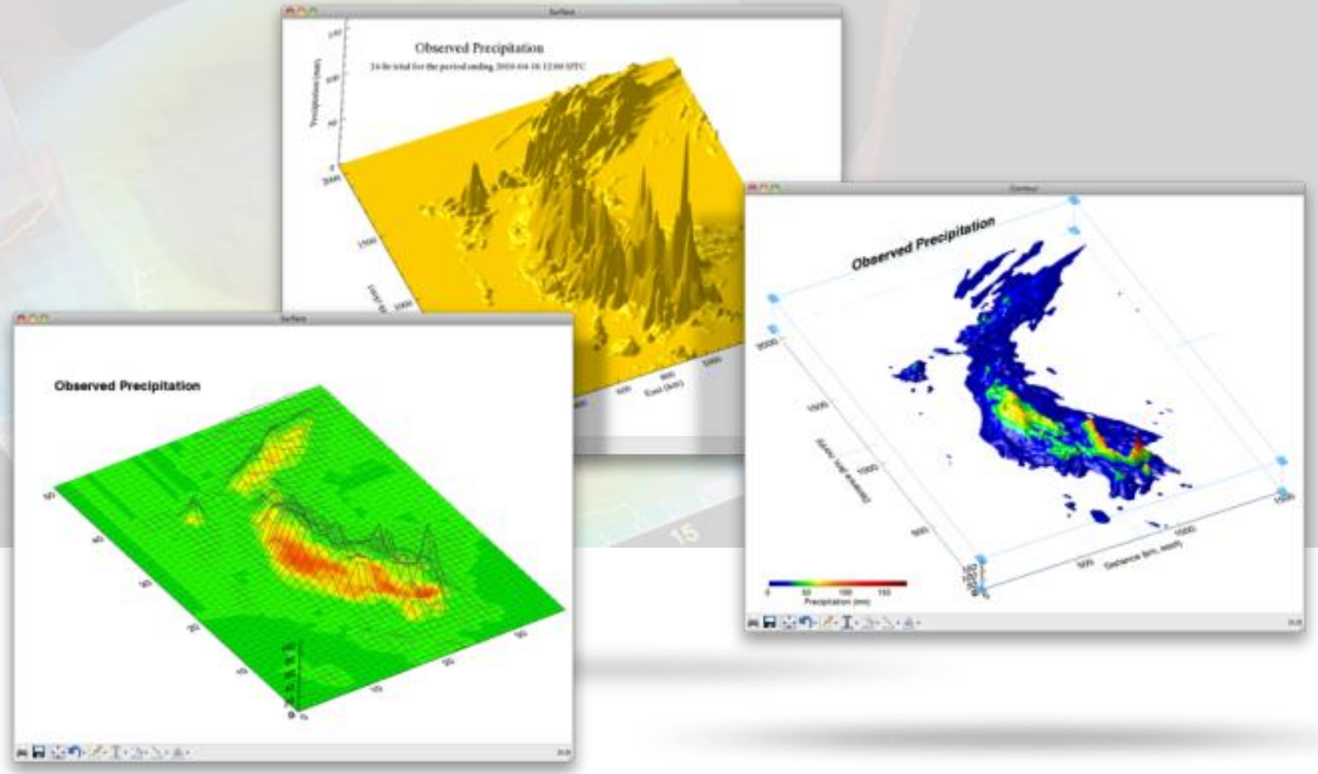
> Language for Analysis, Rules, and Conventions

> Interactive Graphics System

> Development Environment

> Customize ENVI Products with IDL

> **Output File Formats**



Output File Formats

Harris Geospatial Offerings

HARRIS

Geospatial analytics

- **ENVI** – full featured suite of tools, 30+ years of continuous development
- **IDL** – the Interactive Data Language
- **MEGA** – machine learning algorithms
- **GSF** – geospatial analytics in the cloud

Applications for vertical markets

- Research
- Utilities
- Transportation
- Defense & Intelligence

How we deliver

- Desktop applications
- On-premise enterprise deployments
- Hosted solutions and services

ENVI

IDL

GEOSPATIAL SERVICES FRAMEWORK
BRING ANALYTICS AND DATA TOGETHER AT SCALE

HARRIS
GEOSPATIAL SOLUTIONS

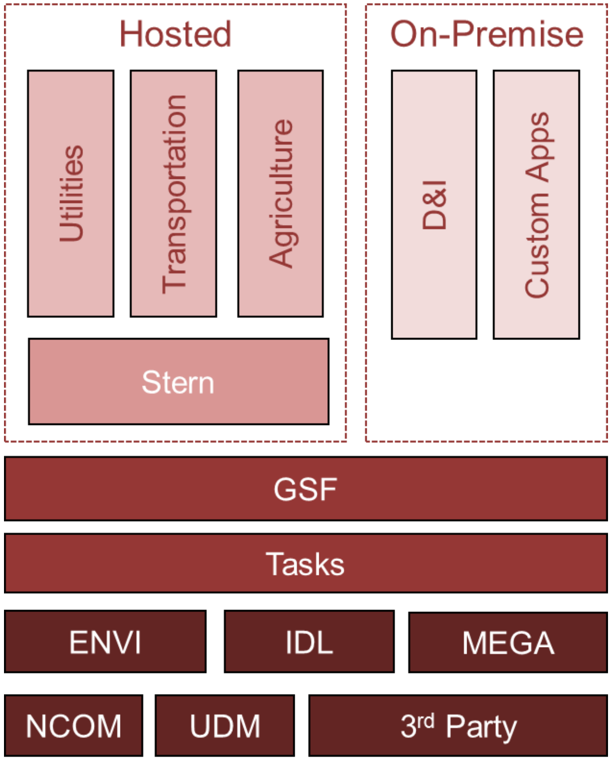
How is it put together?

Organized as fundamental building blocks:

- Analytics
- Data Access
- Visualization
- Applications / UI

How does it fit together?

- **IDL** – the language and library
- **ECF** – ENVI component framework
- **Task Engines** – run analytics at the command line in any environment
- **GSF –** run task engines in a distributed enterprise environment
- **Stern** – host solutions in the cloud



GSF

Geospatial Services Framework

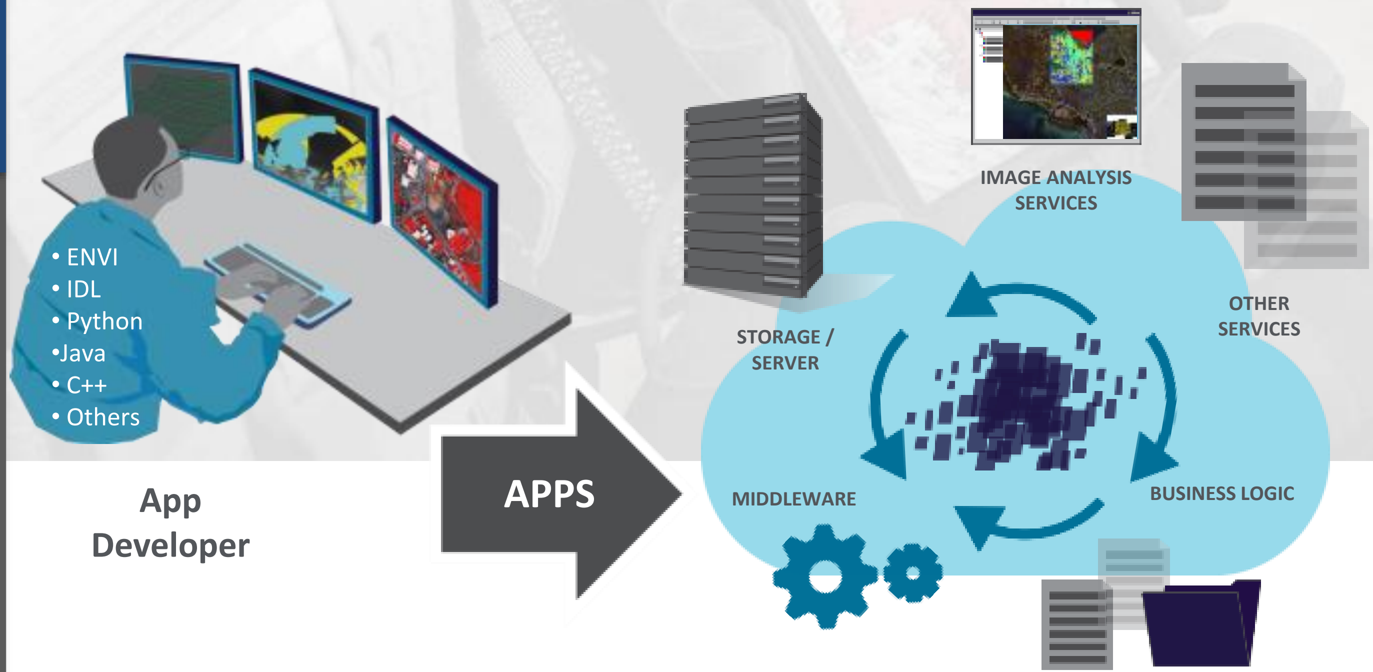
ONLINE, ON-DEMAND

> **Create and publish web deployed image analysis tools**

> Consume IDL or ENVI from mobile, web, and thin clients

> Get imagery where and when you need it

Create Deploy



Create and publish web deployed image analysis tools

GSF

Geospatial Services Framework

ONLINE, ON-DEMAND, GEOSPATIAL AWARENESS

> Create and publish web deployed image analysis tools

> **Consume IDL or ENVI from mobile, web, and thin clients**

> Get imagery where and when you need it

ENVI/IDL Tasks

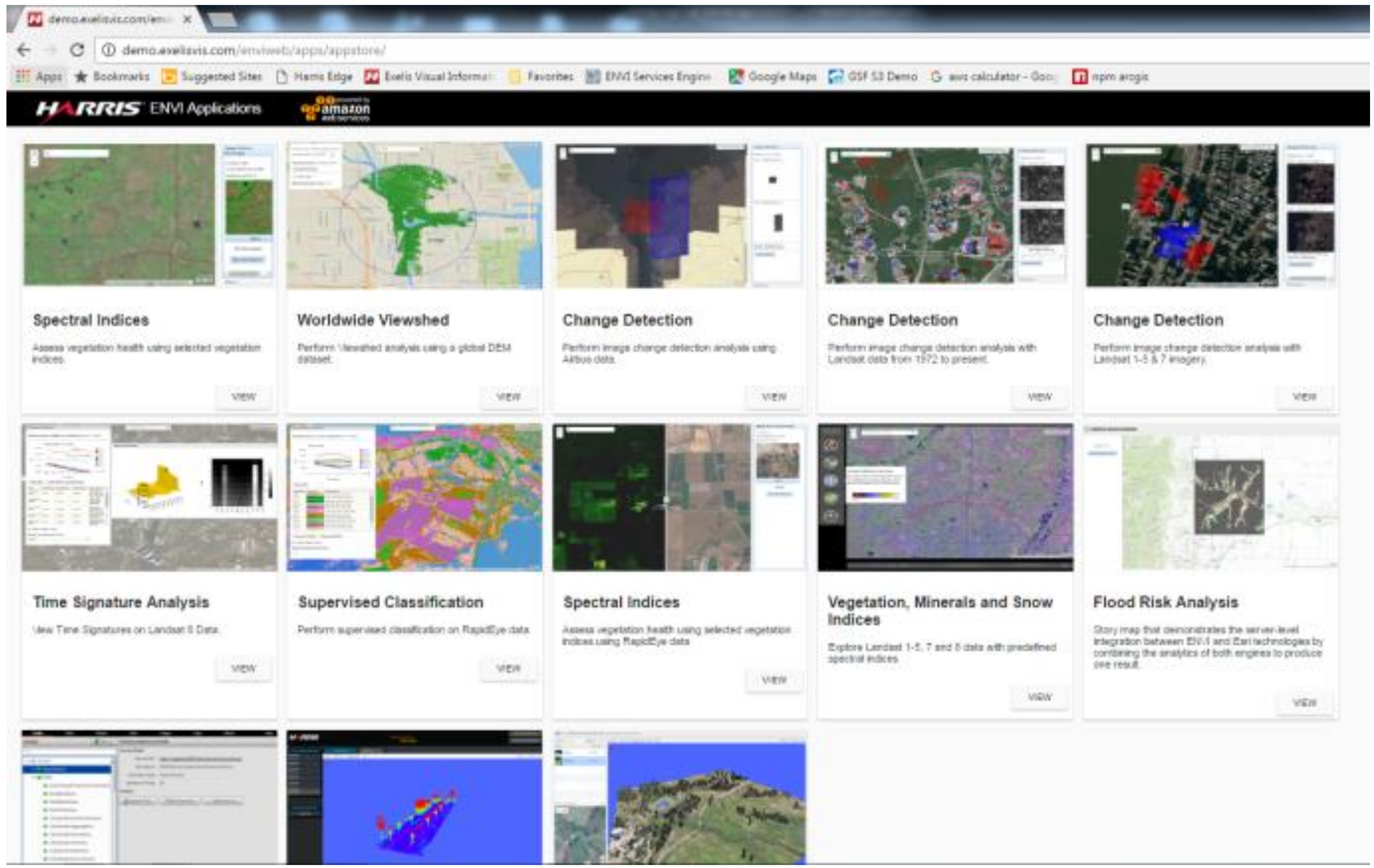
IDL code in a Task

GSF

Geospatial Services Framework

ONLINE, ON-DEMAND, GEOSPATIAL AWARENESS

- > Create and publish web deployed image analysis tools
- > Consume IDL or ENVI from mobile, web, and thin clients
- > **Get imagery/results where and when you need it**



Concepts: Tasks and Engines

The **TASK** is the fundamental unit of analytics.

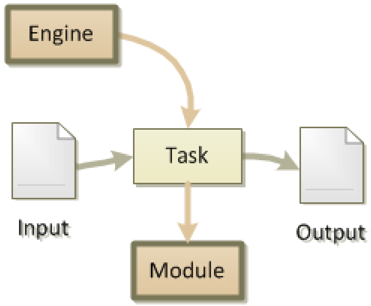
- Accepts data as input
- Performs analytic operations on the data
- Produces data as output

Tasks require an **ENGINE** to run

- Engines require a license

Tasks may require a **MODULE**

- Modules require a license



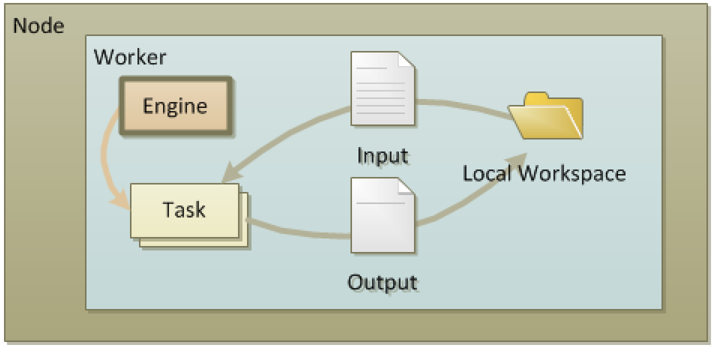
Concepts: Workers and Nodes

A **WORKER** provides a environment that hosts as engine

- Manages engine lifecycle
- Provides **WORKSPACE** for accessing data on input and output
- Pulls work from a shared job queue

A **NODE** is a machine capable of running workers

- May be physical or virtual
- May run one to many workers



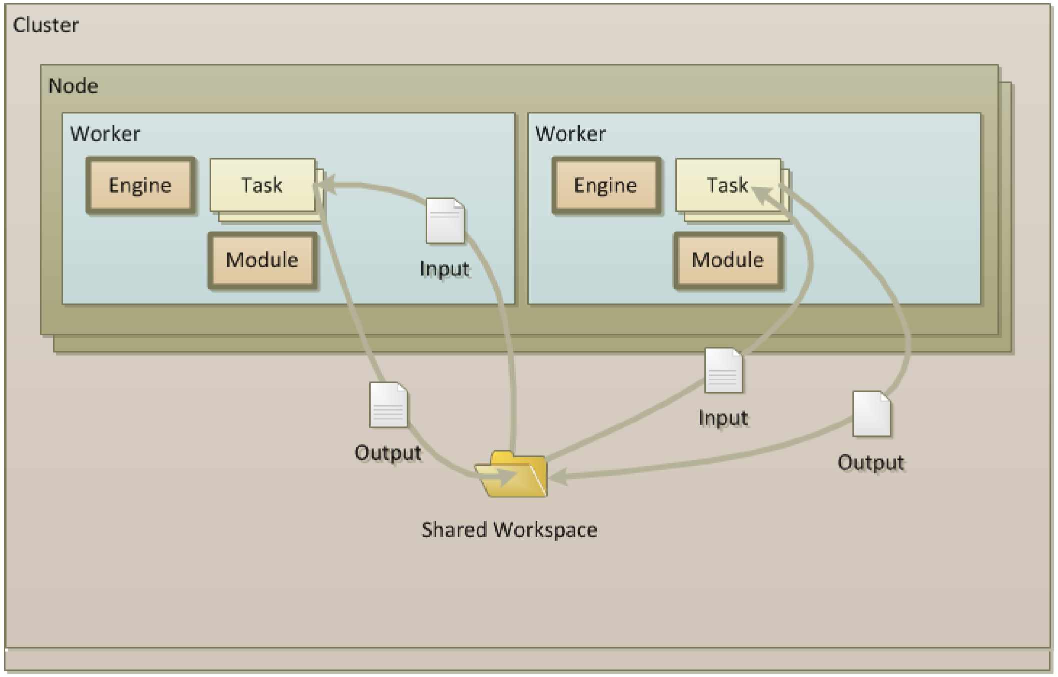
Concepts: Clusters and Shared Workspaces

A **CLUSTER** is a collection of nodes

- May be physical or virtual
- Shares a common job queue
- Uses local workspaces by default

A **SHARED WORKSPACE** is often convenient in a clustered environment

- Each worker can read and write from a common location



Tasks as Analytic Building Blocks

Tasks serve to:

- Creation of new analytic tools
- Support universal deployment (desktop, cloud, distributed computing)
- Provide an easy extension point for customers and partners

Tasks are easily combined:

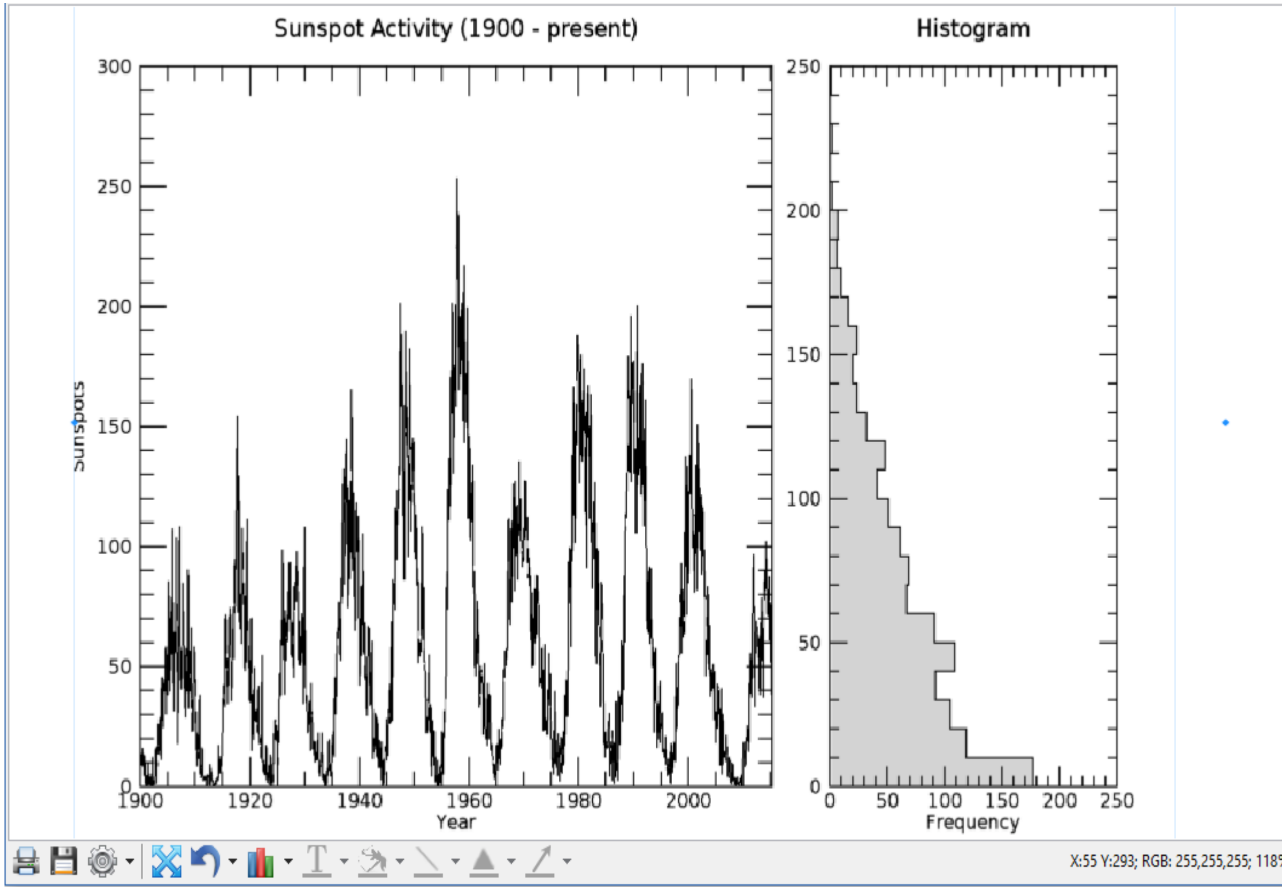
- Chaining
  - Metatasks
  - Workflows
- Custom tasks

203 pre-build ENVI tasks

Unlimited IDL tasks creation

IDLTask: Inputs and outputs Example

YEAR	MON	SSN	DEV
1749	1	58.0	24.1
1749	2	62.6	25.1
1749	3	70.0	26.6
1749	4	55.7	23.6
1749	5	85.0	29.4
1749	6	83.5	29.2
1749	7	94.8	31.1
1749	8	66.3	25.9
1749	9	75.9	27.7
1749	10	75.5	27.7
1749	11	158.6	40.6
1749	12	85.2	29.5
1750	1	73.3	27.3
1750	2	75.9	27.7
1750	3	89.2	30.2
1750	4	88.3	30.0
1750	5	90.0	30.3
1750	6	100.0	32.0
1750	7	85.4	29.5
1750	8	103.0	32.5
1750	9	91.2	30.5
1750	10	65.7	25.7
1750	11	63.3	25.3
1750	12	75.4	27.7
1751	1	70.0	26.6



IDLTask: Creation example

IDL code + JSON module

```
pro sunspot_ise, START_YEAR=startyear, OUTPUT_DIR=outdir

  oURL = IDLnetURL(SSL_VERIFY_HOST=0, SSL_VERIFY_PEER=0)
  cd, outDir
  filename = 'spot_num.txt'
  print, oURL.get(url='https://solarscience.msfc.nasa.gov

  ; Read in the sunspot cycle time series.
  sunspot_ise_read, filename, data

  ; Make a time vector from the year + month values in th
  time = data.year + data.month/12.0

  ; Gather sunspot activity since startyear. WHERE functi
  CALDAT, systime(/julian), Month, Day, currYear
  startyear = fix(startyear) > 1749 < (currYear-1)
  i = where(time ge startyear)

  ; Calculate and display a histogram of the sunspot seri
  ; time interval. Use a binsize of 10 years.
  sunspot_histogram = histogram(data[i].ssn, binsize=10,

  ; Combine the series and histogram in one window using
  p = plot(time[i], data[i].ssn, $
    xstyle=1, $
```

```json
{
    "name": "sunspot_ise",
    "base_class": "IDLTaskFromProcedure",
    "routine": "sunspot_ise",
    "display_name": "IDL Services Engine Demo",
    "description": "Sunspots task example in IDL.",
    "schema": "idltask_1.1",
    "parameters": [
            {
                "name": "START_YEAR",
                "display_name": "Start Year",
                "type": "integer",
                "direction": "input",
                "required": true
            },
            {
                "name": "OUTPUT_DIR",
                "display_name": "Output Folder",
                "type": "string",
                "direction": "output",
                "required": true
            }
        ]
}
```

IDLTask: Calling example

IDL classic call:

```
sunspot_ise, START_YEAR=1900, OUTPUT_DIR='C:\Resources\IDL_TEST\'
```

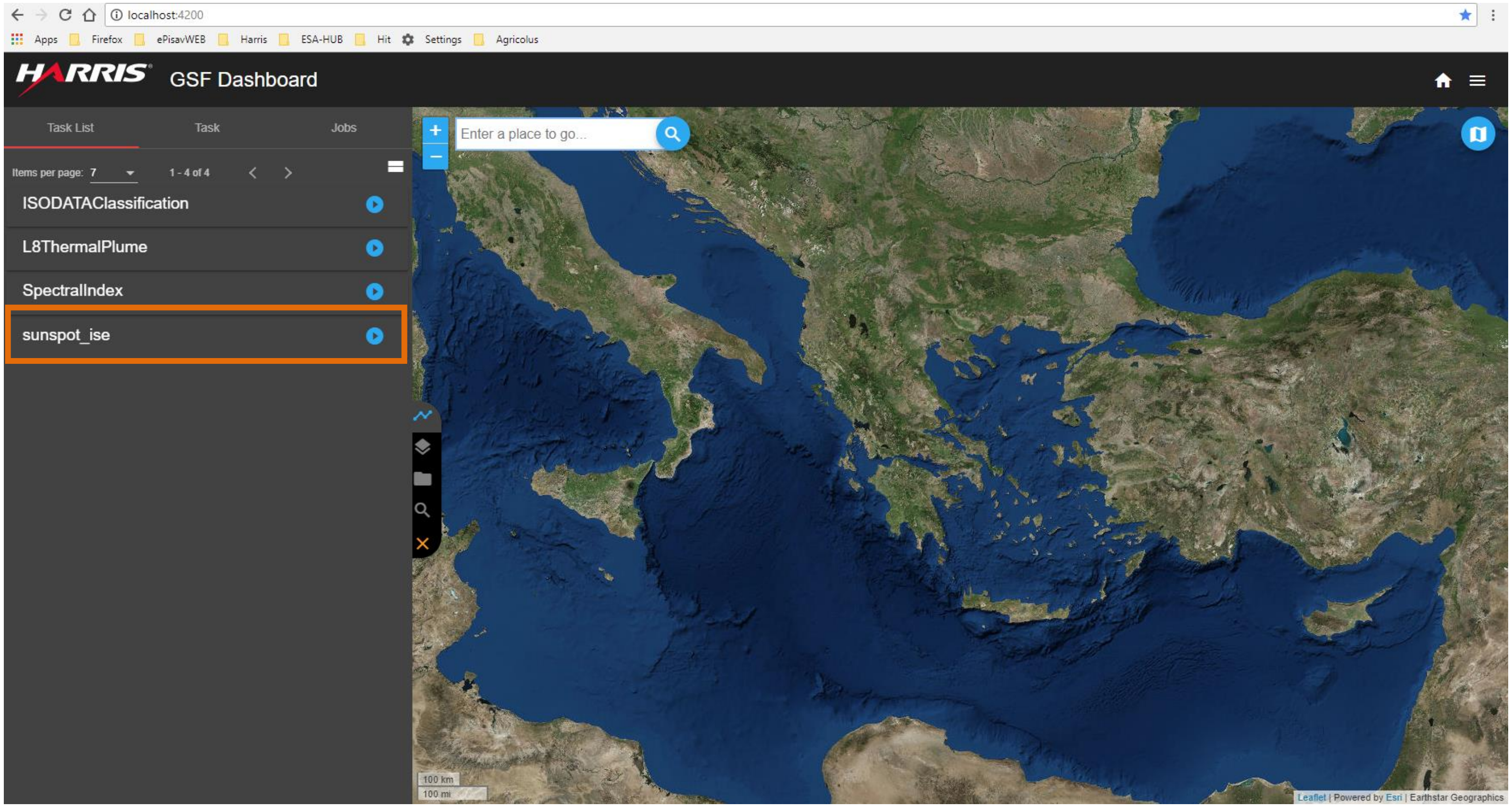
IDL Task method call:

```
task = IDLTask('sunspot_ise')
task.START_YEAR = 1900
task.OUTPUT_DIR = 'C:\Resources\IDL87_PROJECTS\ISE_Sunspot
task.Execute, ERROR=err
```

HTTP, (GSF + IDL Engine) call:

http://13.73.142.221:9191/ese/services/IDL/sunspot_ise/SubmitJob?START_YEAR="1900"&OUTPUT_DIR="C:\Resources\IDL87_PROJECTS\ISE_Sunspot"

WEB Application call !



Metatasks

Metatasks chain small atomic tasks into larger analytic units

- Component tasks may be standard ENVItasks, custom IDLTasks or any combination that share common data types
- Metatasks behave like any other task, and can be chained into larger metatasks

Two types of metatasks exist in the stack:

- IDL/ENVI metatasks – run on a single processing node
- GSF metatasks – parallel component tasks can run independently (in parallel) on different nodes

IDL metatasks run on desktop, GSF

GSF metatasks run only on GSF

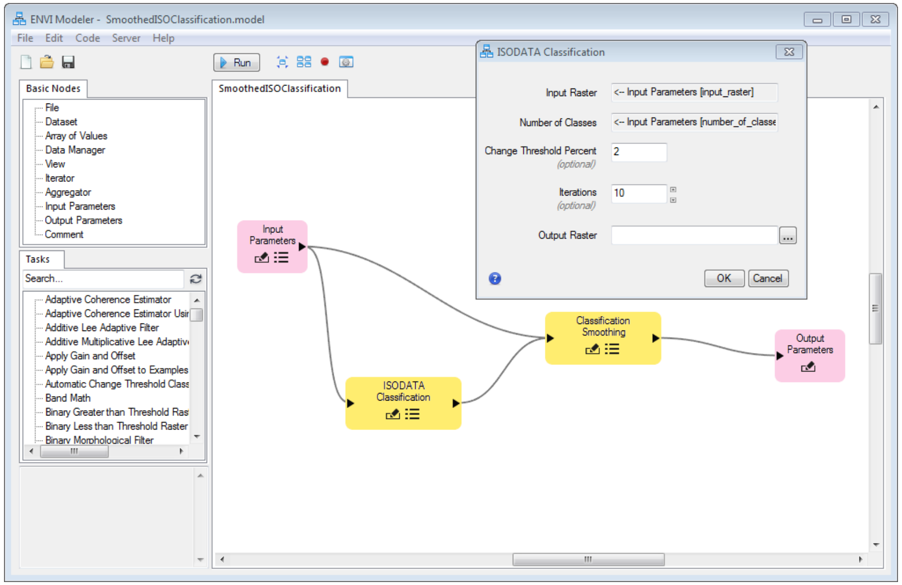

```
{
  "base_class": "ENVIMETATASK",
  "display_name": "SmoothedISOClassification",
  "description": "This is an ENVI Metatask",
  "schema": "envitask_3.2",
  "revision": "1.0.0",
  "commute_on_downsample": "Unknown",
  "commute_on_subset": "Unknown",
  "name": "SmoothedISOClassification",
  "parameters": [
    { "name": "INPUT_RASTER", ... },
    { "name": "NUMBER_OF_CLASSES", ... },
    {  "name": "KERNEL_SIZE", ... },
    {  "name": "OUTPUT_RASTER", ... },
    {
      "name": "DAG",
      "type": "ENVIMETATASKDAG",
      "direction": "INPUT",
      "required": true,
      "description": "This is the graph that describes the metatask.",
      "hidden": true,
      "default": {
        "task_1": {
          "name": "ISODATAClassification",
          "external_input": {
            "input_raster": "INPUT_RASTER",
            "number_of_classes": "NUMBER_OF_CLASSES"
          },
          "internal_input": {
          },
          "static_input": {
          },
          "output": {
          }
        },
        "task_2": {
          "name": "ClassificationSmoothing",
          "external_input": {
            "kernel_size": "KERNEL_SIZE"
          },
          "internal_input": {
            "input_raster": "task_1.output_raster"
          },
          "static_input": {
          },
          "output": {
            "output_raster": "OUTPUT_RASTER"
          }
        }
      },
      "allow_null": false
    }
  ]
}
```

ENVI Modeler

Both ENVI , IDL and GSF tasks can be composed by hand in a text editor.

Much more conveniently, Tasks can be composed using the ENVI Modeler

- Compose and test in real time on ENVI Desktop
- Run your task with automatic UI creation
- Use code generation to emit a metatask, which can be deployed on the desktop, GSF



The Evolution of Harris Geospatial

HARRIS®

Historically, we have been a provider of geospatial analytics

- Desktop/workstation-centric
- Emphasis on tools
- Application of tools to solve problems has been left to the user

Two major shifts in our market are under way

- Customers want to run analytics in the cloud rather than on the desktop
- Customers value **answers** to questions, rather than tools and data

In response, our focus has shifted to delivering solutions:

- Built from a common library of analytics building blocks
- Customized to the needs of users in vertical markets

Solutions are delivered as either:

- Hosted applications in the public cloud
- On-premise applications

Our cloud-native platform is **GSF** (Geospatial Services Framework).

Our SaaS offering is **Stern**.

Scalability

Tasks are the atomic unit of work, and the key to scalability

- By definition, a task can execute independently

Large tasks and monolithic tasks require ever-greater monolithic compute resources

- That is, they only scale **vertically**
- Vertical scalability is inherently limiting and prohibitively expensive

Small atomic tasks can run in parallel

- That is, they scale **horizontally**
- Horizontal scalability is unlimited and relatively cheap

Design principles for Tasks

- Keep tasks small and focused: Do one thing, do it well
- Build complex tasks and workflows from small building blocks
- This supports both reuse and scalability

Scaling Horizontally

HARRIS

Horizontal scaling requires orchestration

Orchestration requires:

- **Job** – request to run a task
- **Queue** – list of pending job requests
- **Job Manager** – manages a queue of jobs and orchestrates execution on a pool of execution engines
- GSF provides orchestration for Task Engines

GSF supports a cluster model

- Nodes run task engines
- Nodes can be specialized (routing)
- Nodes can be added and removed while the system is live

The Human Bottleneck

- GSF is flexible and horizontally scalable
- But, each instance of GSF has to be installed and configured by a human
- Each node in the GSF cluster must also be manually configured
- While GSF clusters can grow and shrink, this is also manual process

docker kubernetes

Bigdata Processing

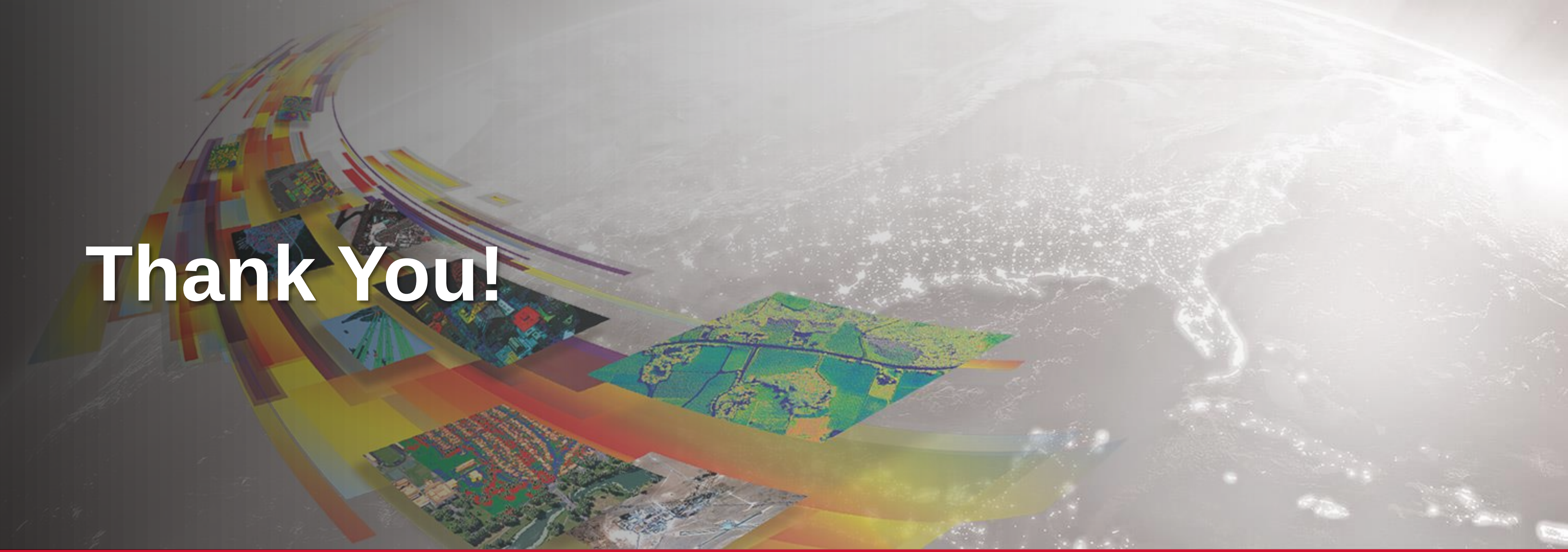
Since a job is an encapsulated and atomic processing task, its execution cannot be split on different workers or nodes (including IDL child processes). GSF was built to support the concept of scaling out on cloud infrastructures such as AWS and Azure to process more datasets concurrently.

Work in parallel using GSF task framework model:

At a high level, this model should work in ENVI or as a JavaScript task in GSF. Such a metatask would then look as follows:

- Submit a Job to split: Raster in → split to multiple chunks of data → wait for split task to finish
- Submit a job for each chunk of data (GSF will copy the chunks to each node for processing) → wait for all jobs to finish
- Submit a job to merge: Results from all chunks → merge results → wait for merge
- Return merged results

Questions & Answers



www.harrisgeospatial.com

www.facebook.com/HarrisGeospatialSolutions

https://twitter.com/GeoByHarris

YouTube www.youtube.com/user/ExelisVis

Stefano.Gagliano@harris.com

Download and test GSF!