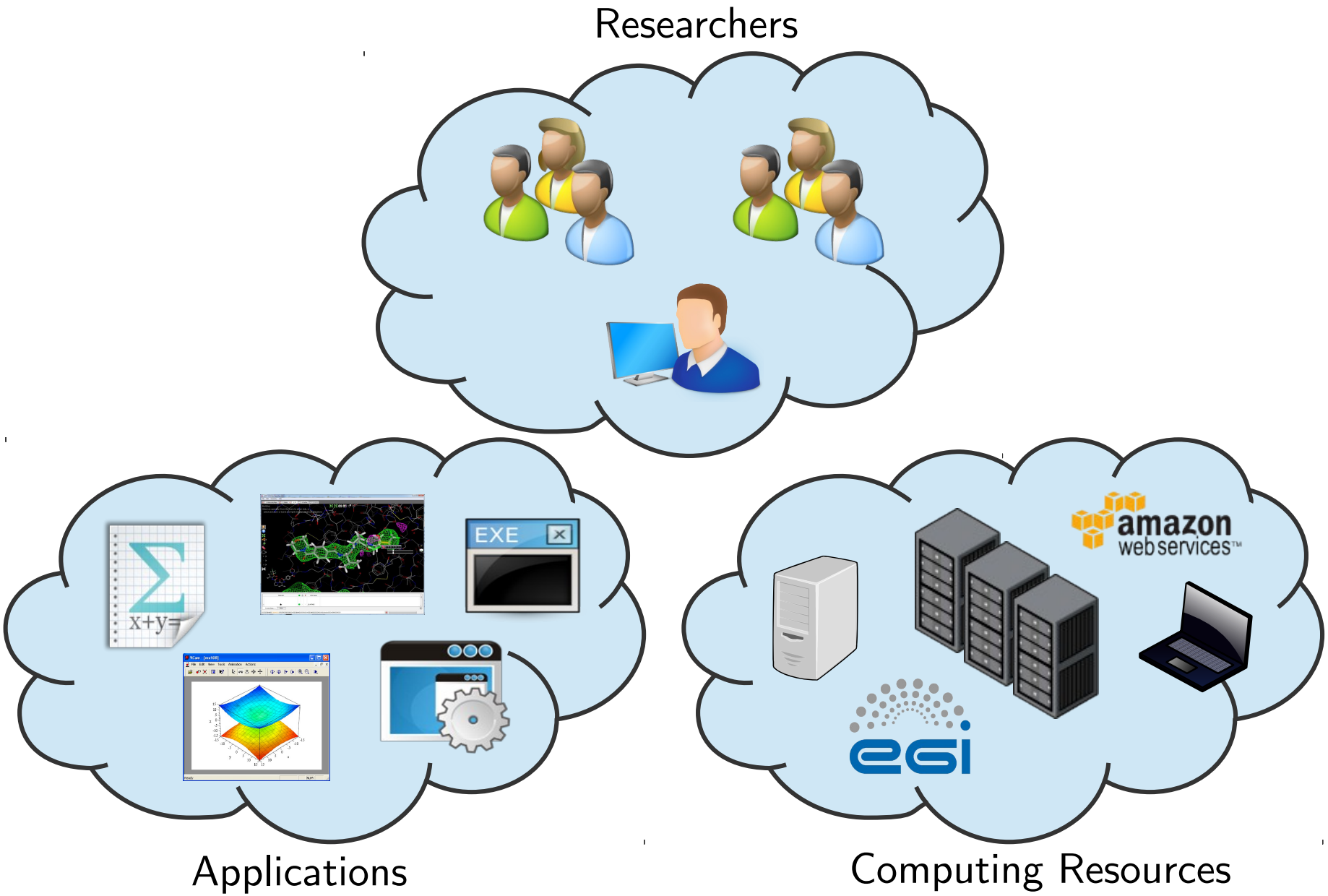
14th International Symposium on Parallel and Distributed Computing

A Web-Based Platform for Publication and Distributed Execution of Computing Applications

Oleg Sukhoroslov, Sergey Volkov, Alexander Afanasiev

Institute for Information Transmission Problems (Moscow, Russia)

Motivation


Researchers
EXE
amazon
web services
egi
Applications
Computing Resources

Challenges

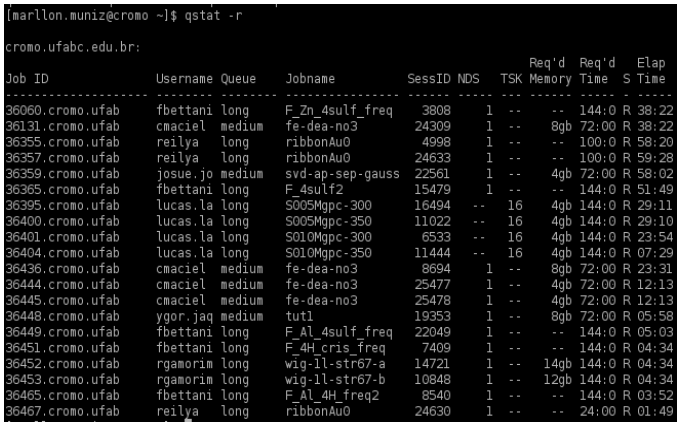
- Convenient access to computational applications
- Execution of applications on heterogeneous distributed computing resources
- Automation of workflows involving multiple applications
- Sharing applications/workflows with colleagues

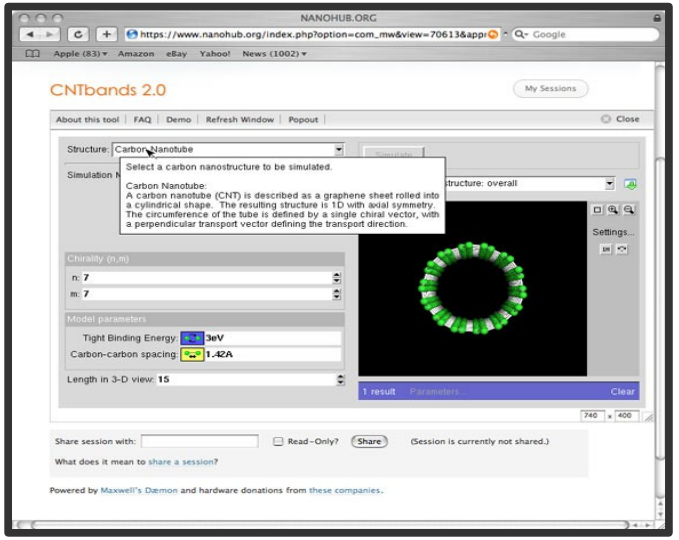
Current Solutions

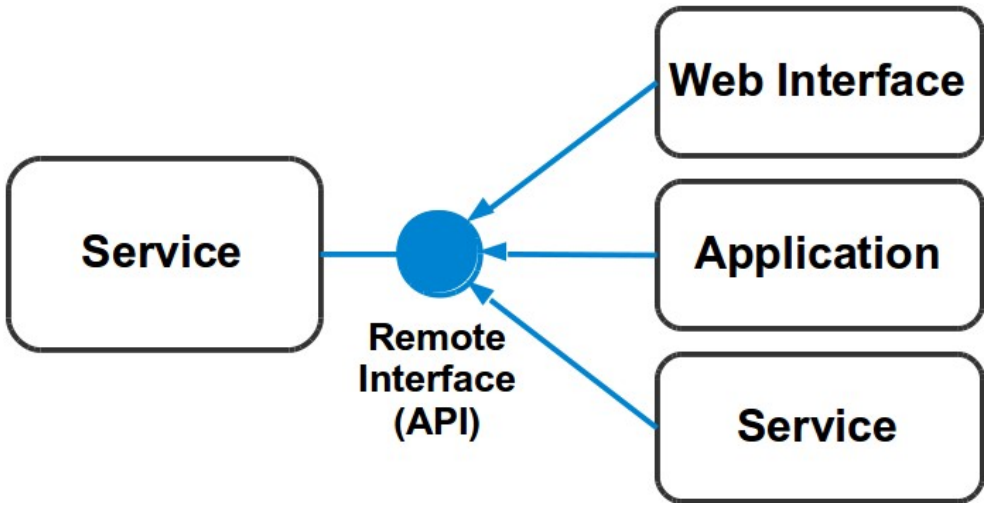
	Hosted vs Standalone	Arbitrary Resources	Application Sharing	Application Composition	Remote API
Grid Middleware					
User-level Toolkits / Workflow Systems					
Scientific Gateways					
Web Service Toolkits					

Sharing a Scientific Application

- Publish a source code
- Send an executable
- Create a web-based interface
- Run a web service







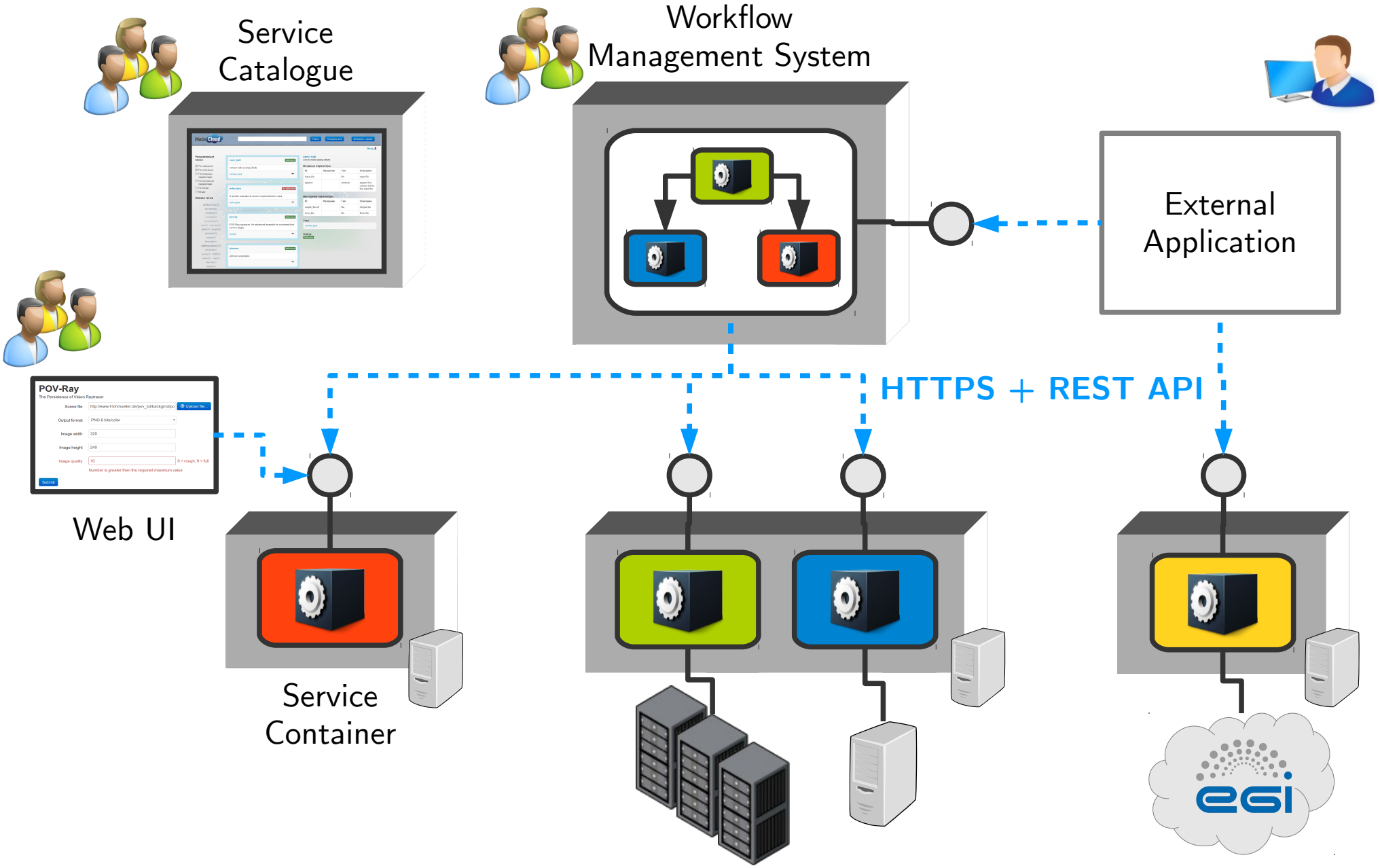
Scientific Application as a Service

- Software as a Service (SaaS)
- No need to install software and deal with computing resources
- Centralized maintenance and accelerated feature delivery
- Application composition and integration with third-party tools
- Collaboration
- Publication and reproducibility

MathCloud (2009-2013)

- Software toolkit for building, deployment, discovery and composition of computational web services
- Based on the unified web service interface
  - Follows REST (Representational state transfer) architectural style
- Main components
  - Service Runtime Environment (Container)
  - Service Catalogue
  - Workflow Management System (WfMS)
  - Security Mechanism
  - Client Interfaces

MathCloud Architecture



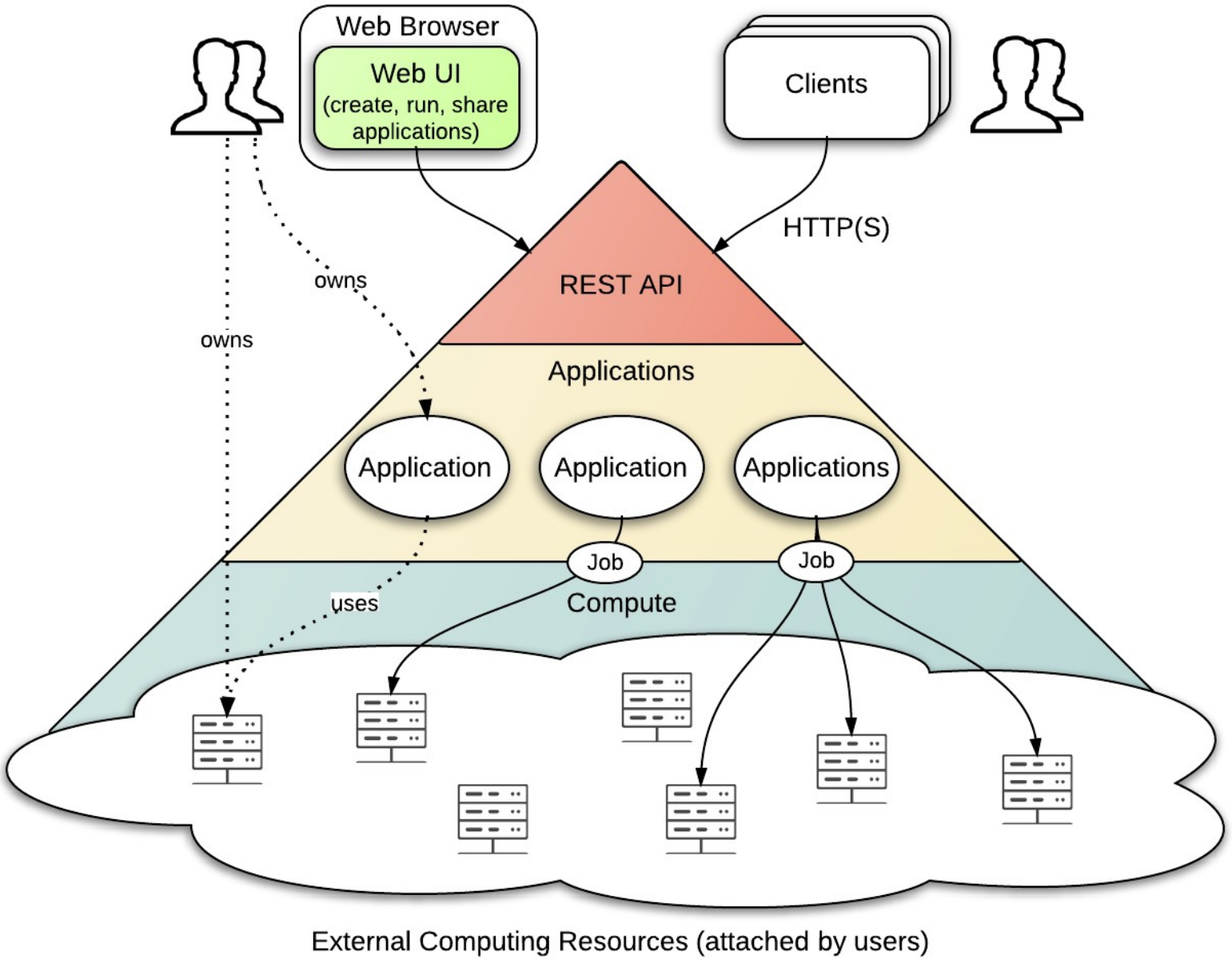
Problems

- Lack of convenient infrastructure to host services
- Sharing an application implies sharing a resource
- Service user cannot override the resource

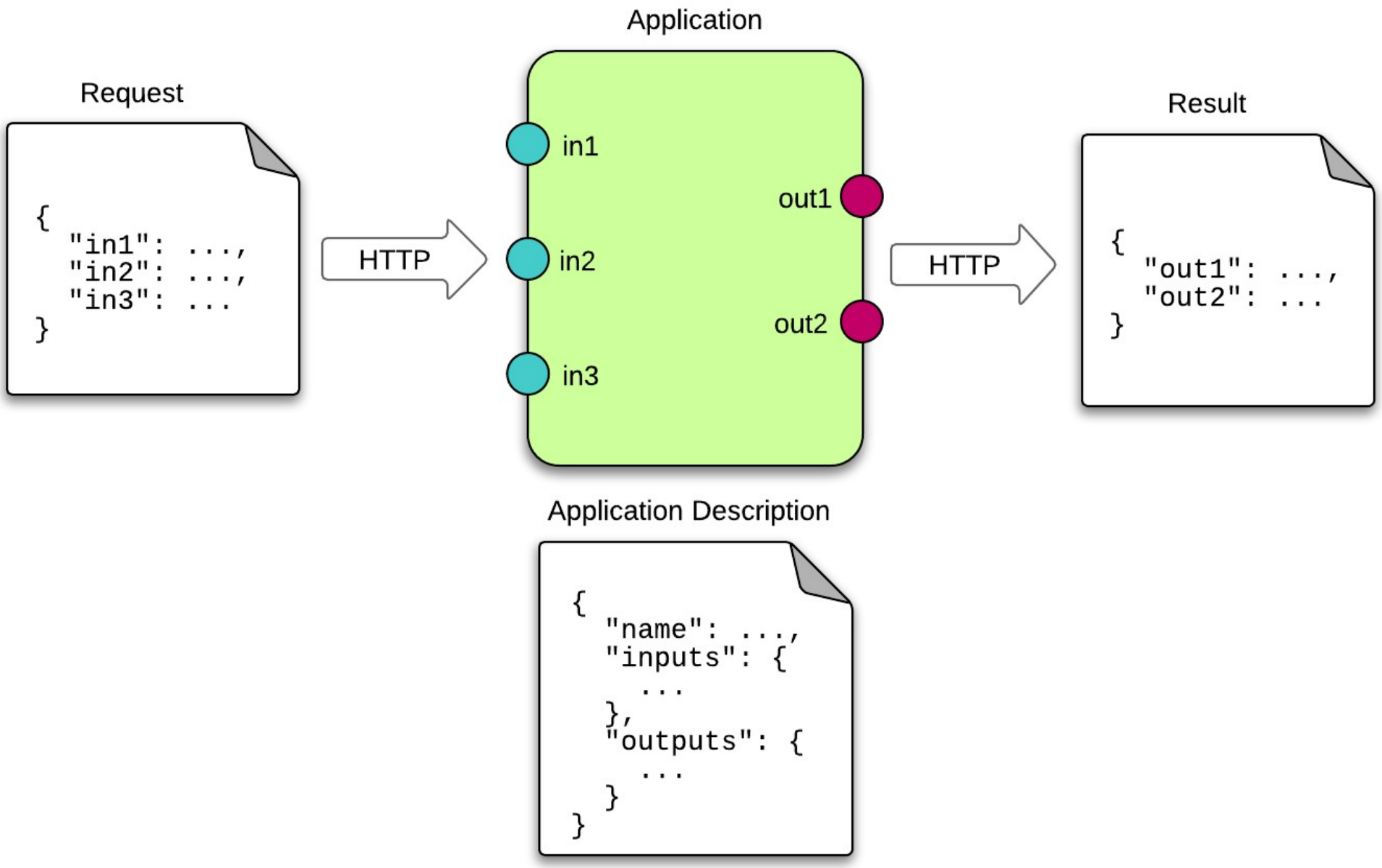
Everest (2014-...)

- Leverage cloud computing models to implement a web-based platform supporting
  - Describing and hosting computational applications as services
  - Binding applications to external computing resources
  - Running applications on arbitrary sets of resources
  - Sharing applications and resources with other users
- Platform as a Service (PaaS)
  - Accessible via web browser and REST API
  - No installation is required
- Combination of existing approaches + PaaS
  - Uniform REST interface for accessing applications
  - Web UI for application description
  - Automatic generation of web UI for application invocation

Everest Architecture



Application: Interface



POV-Ray: Parameters

POV-Ray

Edit

About | Parameters | Submit Job

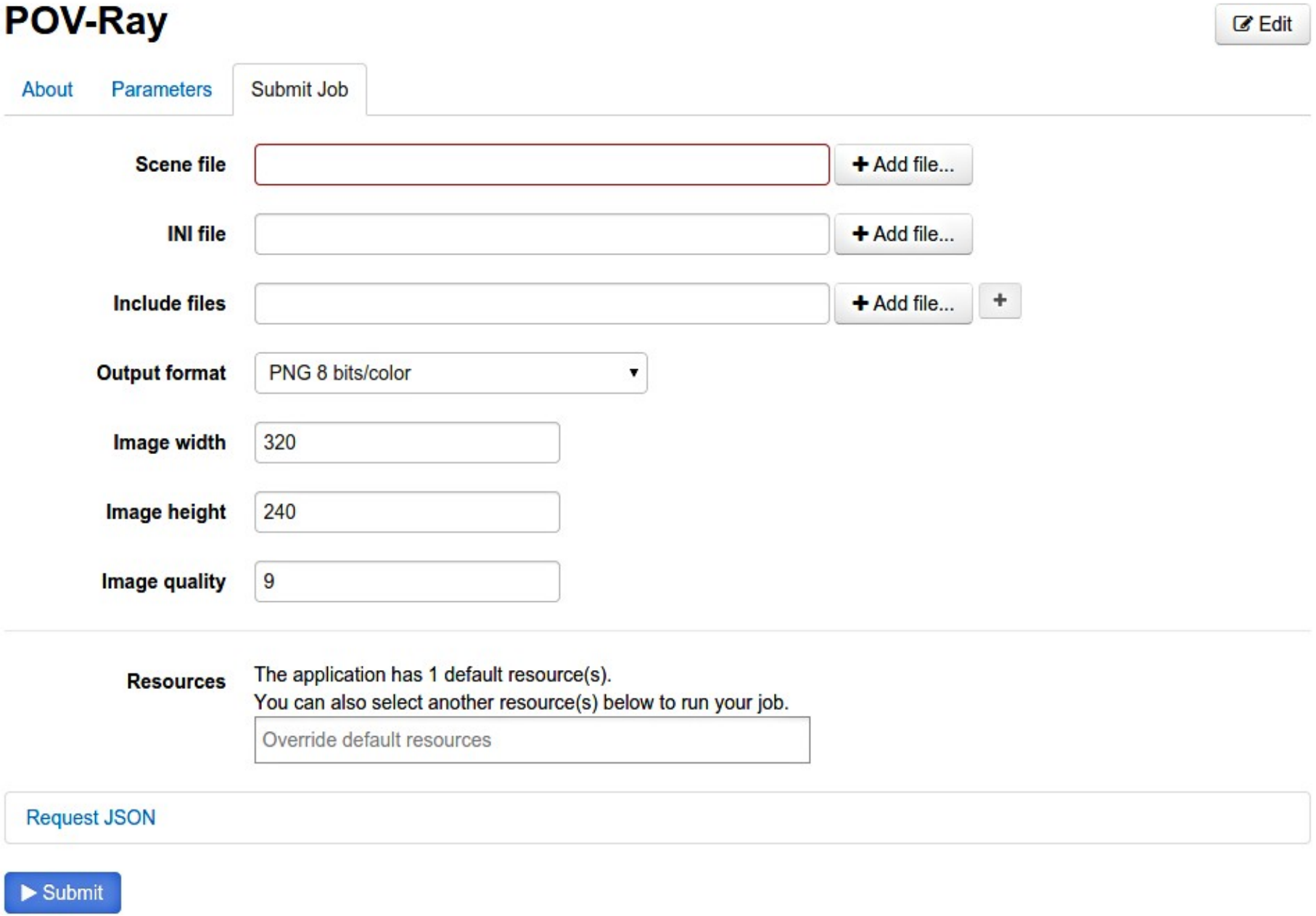
Inputs

	Title	Name	Type	Values	Default	Description
✔	Scene file	scene	URI			
	INI file	ini	URI			
	Include files	includes	array [URI]			
	Output format	format	string	C N8 N16 P T	N8	
	Image width	width	integer	[1, 2048]	320	
	Image height	height	integer	[1, 2048]	240	
	Image quality	quality	integer	[0, 11]	9	

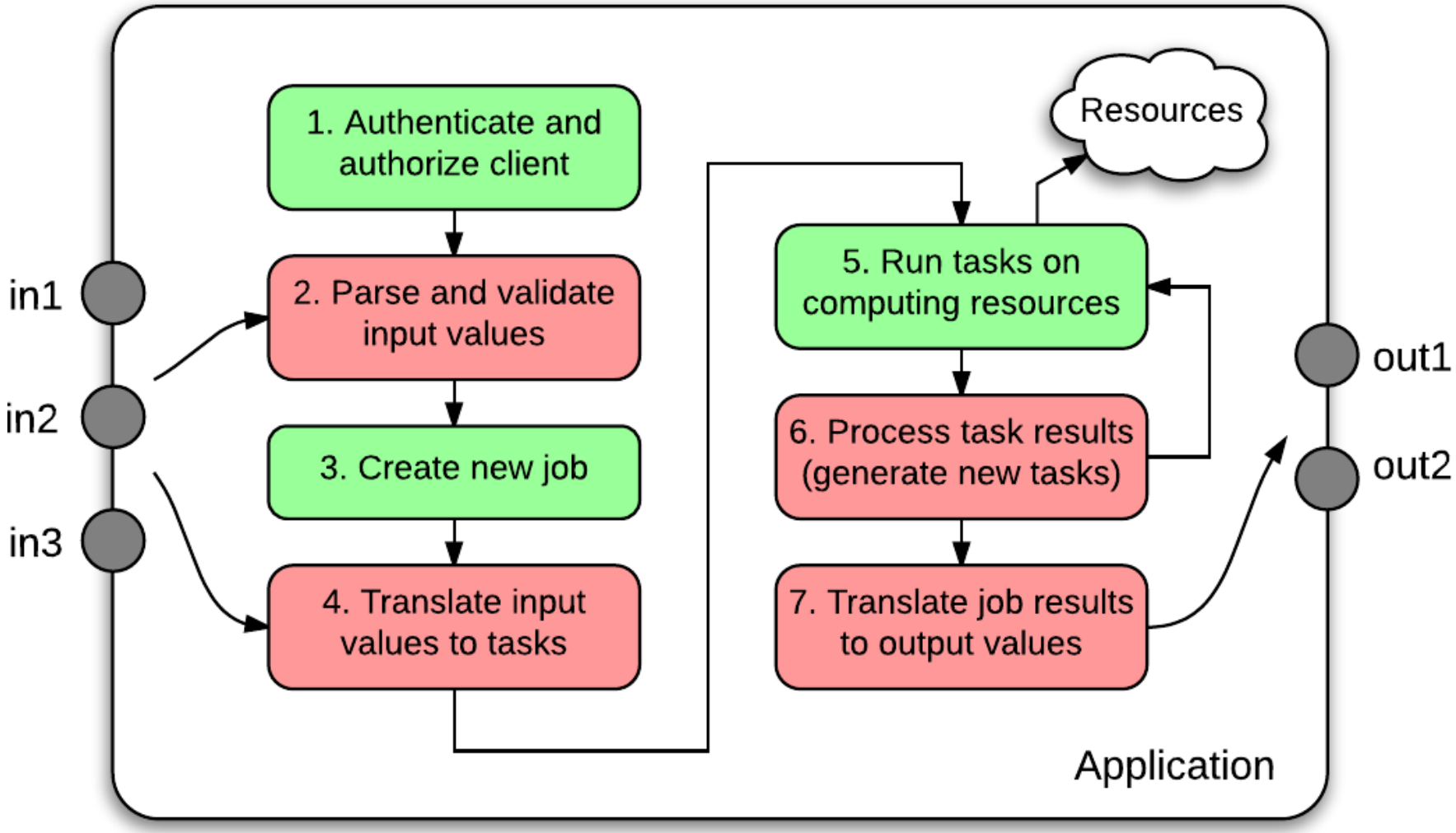
Outputs

	Title	Name	Type	Description
✔	Output image	image	URI	
✔	POV-Ray log	log	URI	

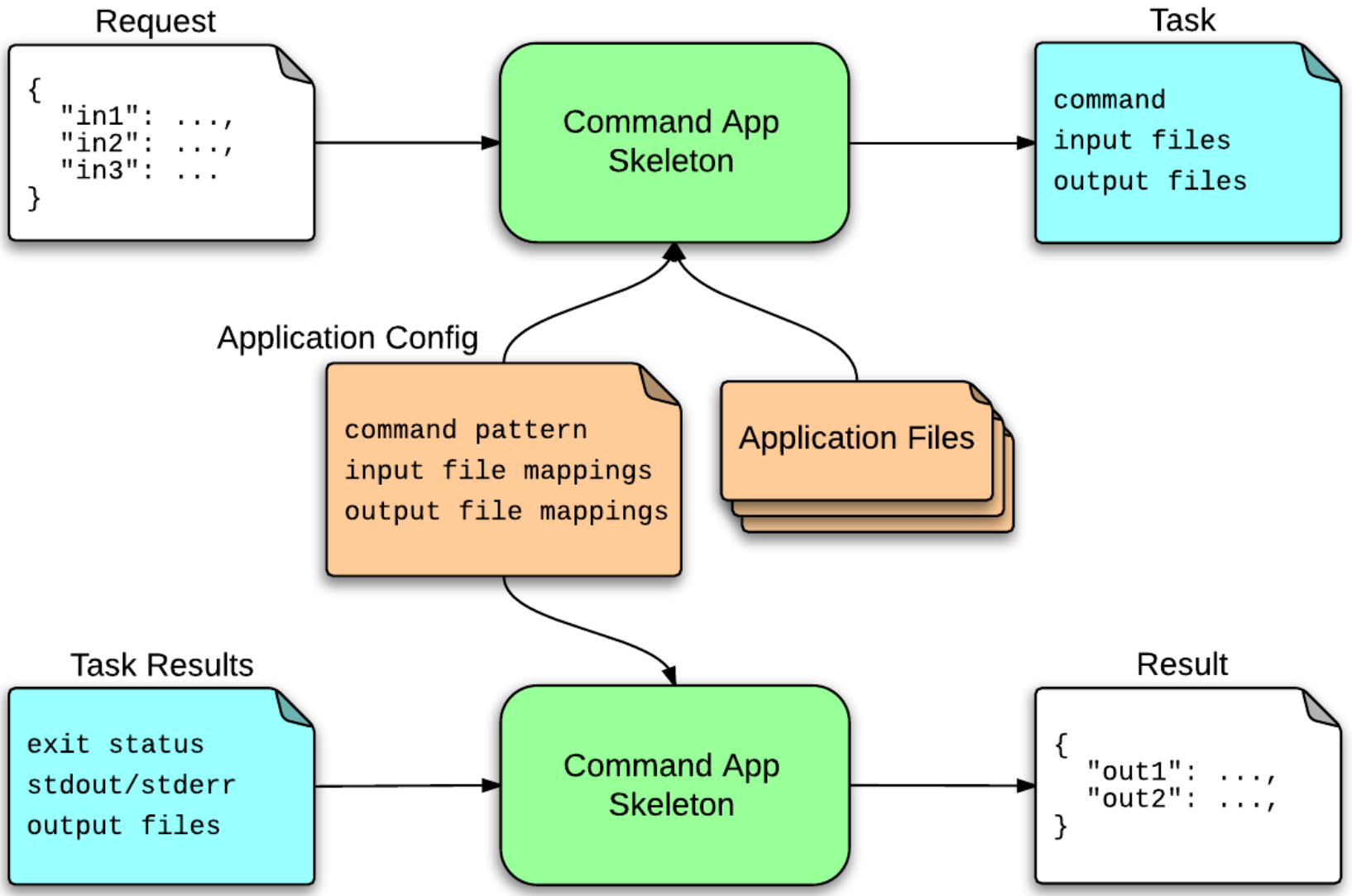
POV-Ray: Submit Form



Application: Implementation



Command Application Skeleton



POV-Ray: Configuration

POV-Ray

Metadata | Inputs | Outputs | **Configuration** | Files | Resources | Access

Command

```
./povray_run.sh +Iscene.pov +F${format} +W${width} +H${height} +Q${quality} -D +A -Oimage
```

Refer to input values as ${param}

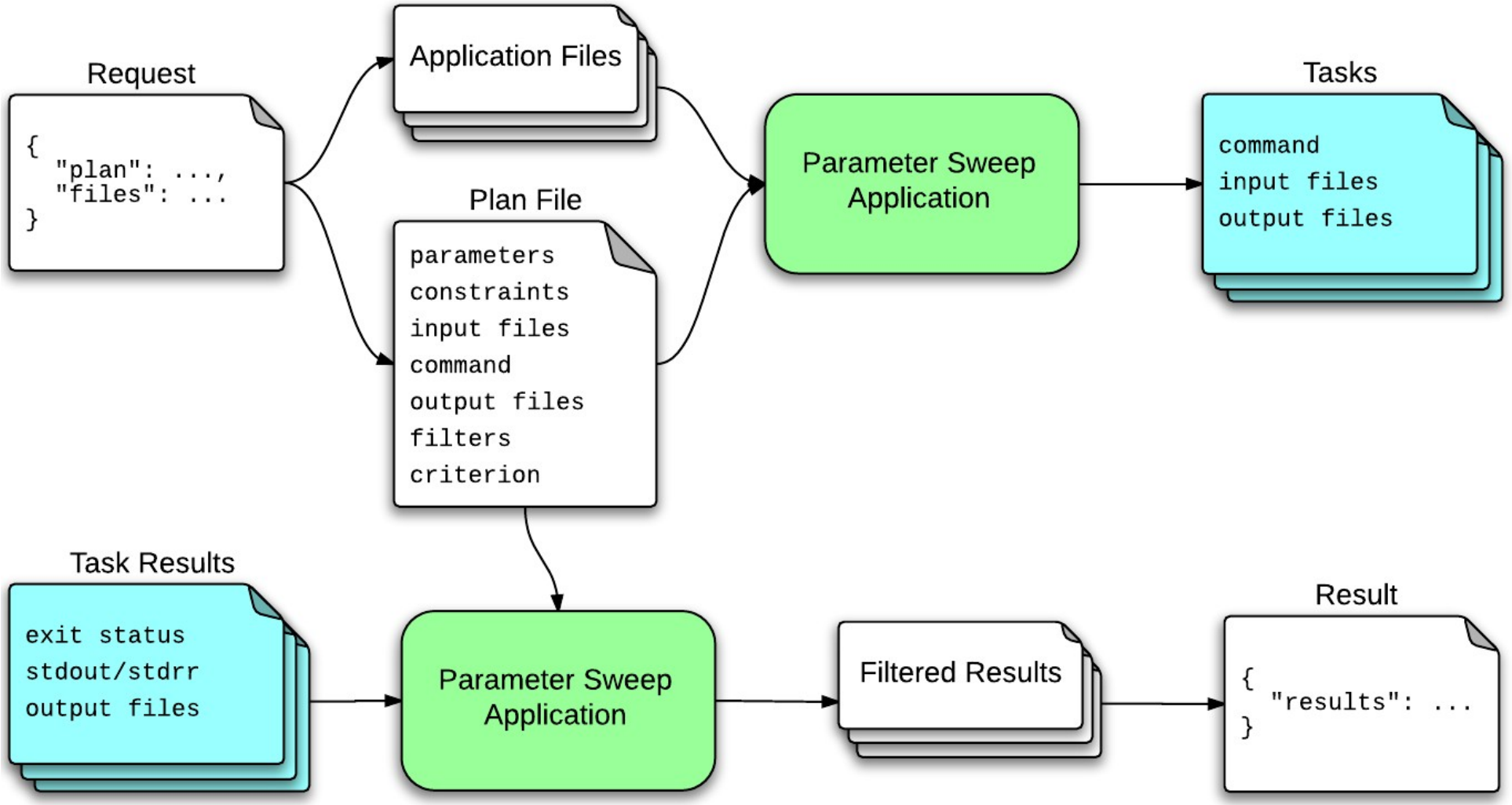
Input Mappings

Input	File	Pattern	
scene	scene.pov		🗑
ini	povray.ini		🗑
			+

Output Mappings

Output	File	Pattern	
image	image.*		🗑
log	stderr		🗑
			+

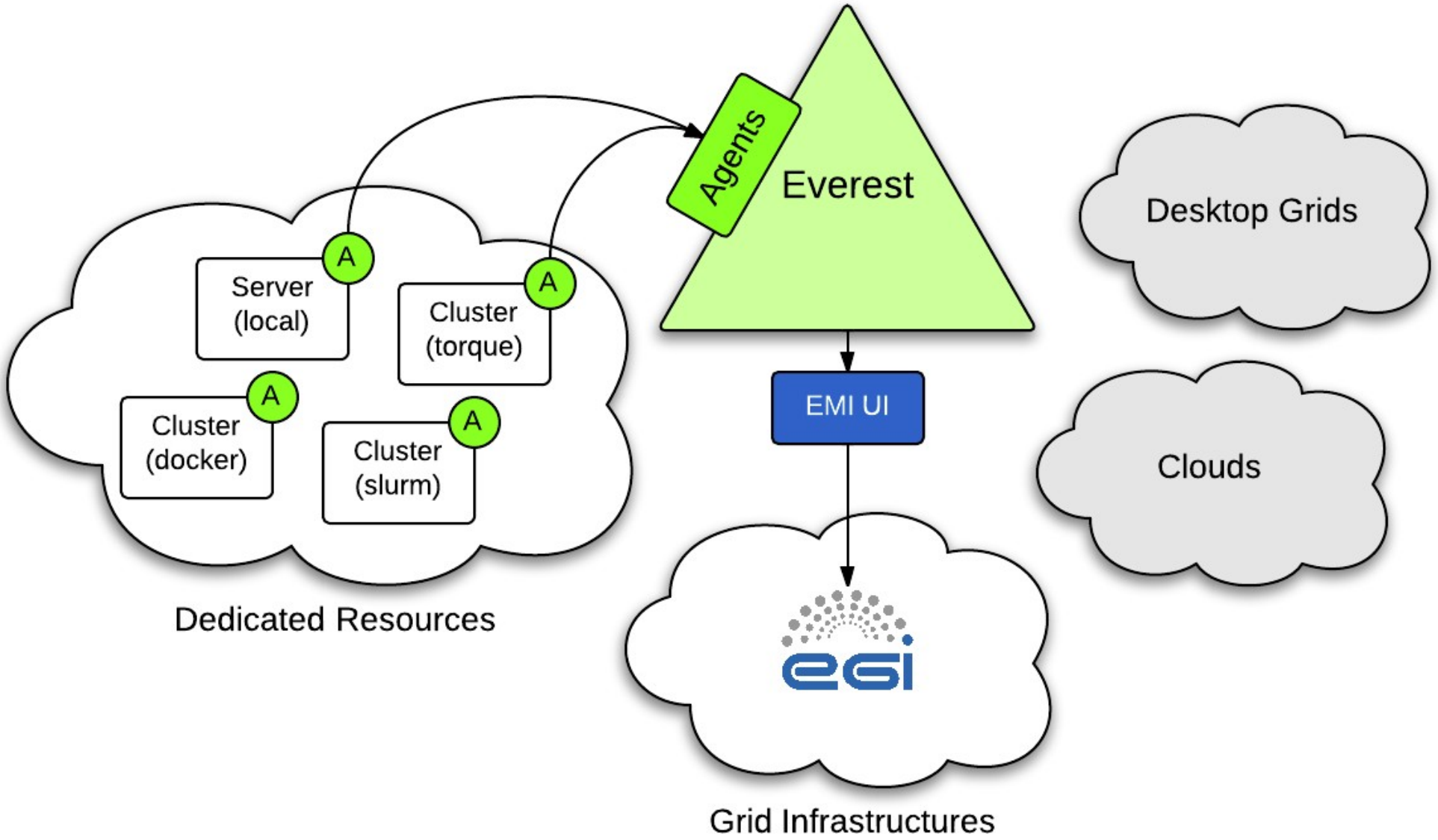
Parameter Sweep Application



Example: Virtual Screening

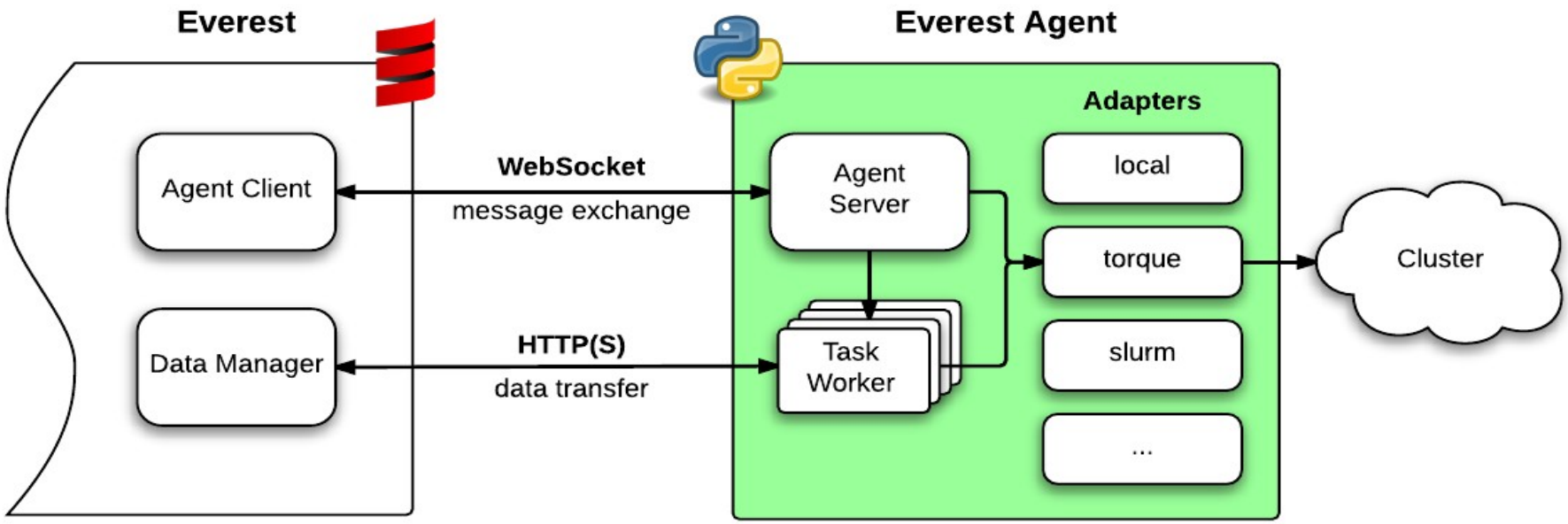
```
parameter n from 1 to 100 step 1

input_files @run.sh vina write_score.py protein.pdbqt
input_files ligand${n}.pdbqt config.txt

command ./run.sh

output_files ligand${n}_out.pdbqt log.txt @score

criterion min $affinity
```

Integration with Computing Resources

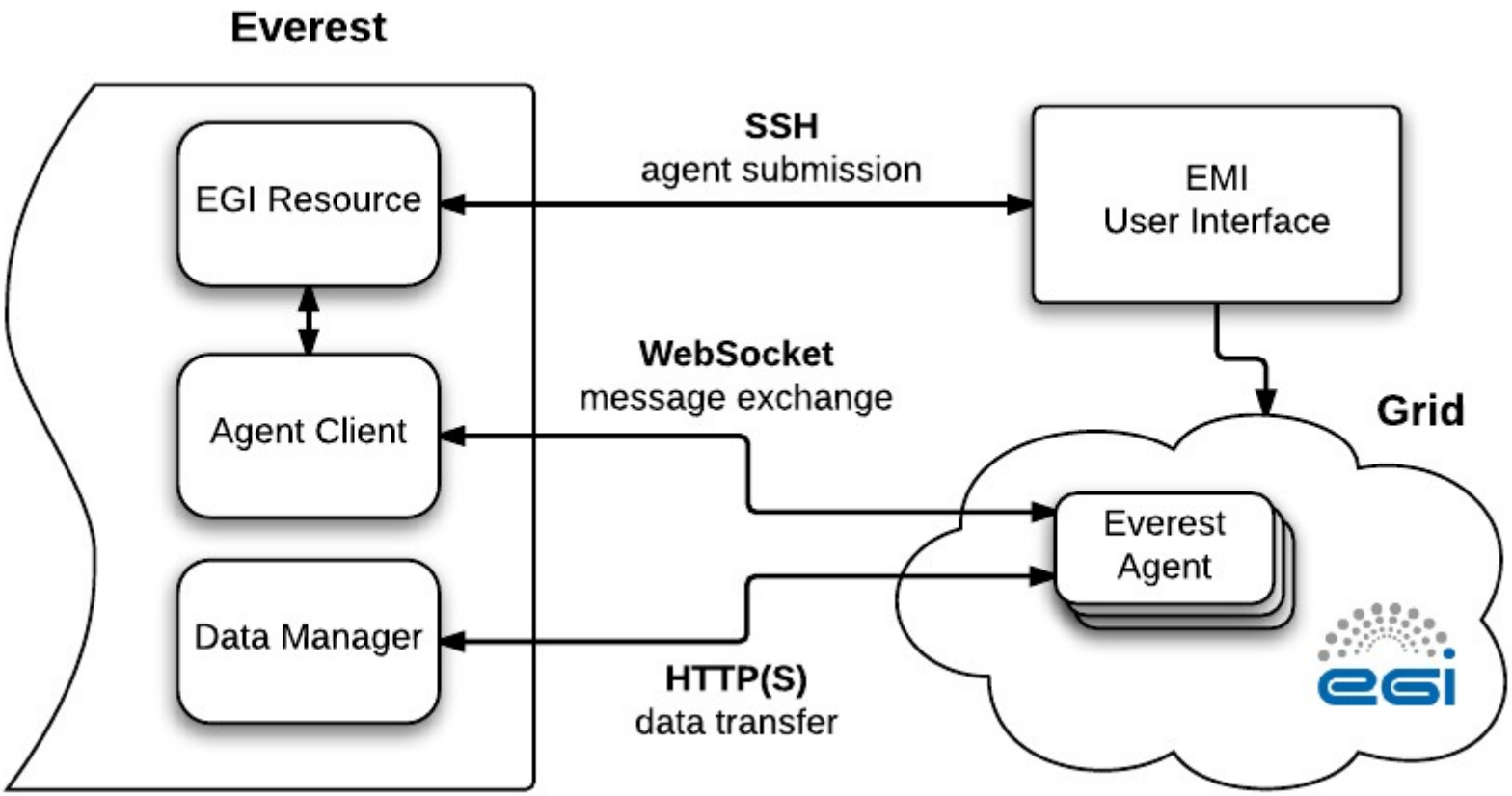


Everest Agent

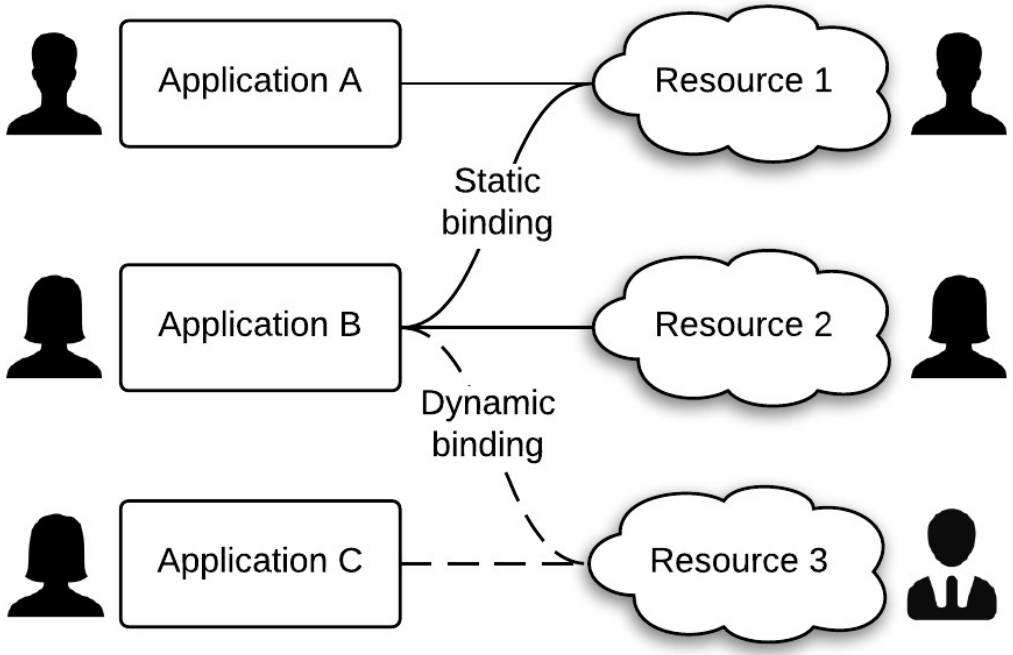
- A mediator between the resource and the platform
- Supporting servers, clusters and resources behind a firewall
- Security mechanisms: white list, execution of tasks in Docker containers
- Open Source: https://gitlab.com/everest/agent/

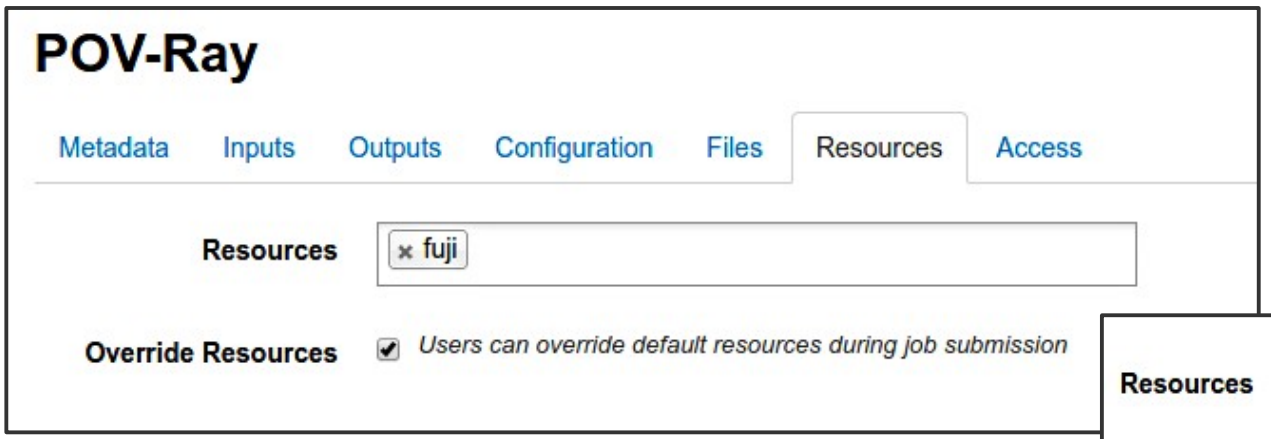


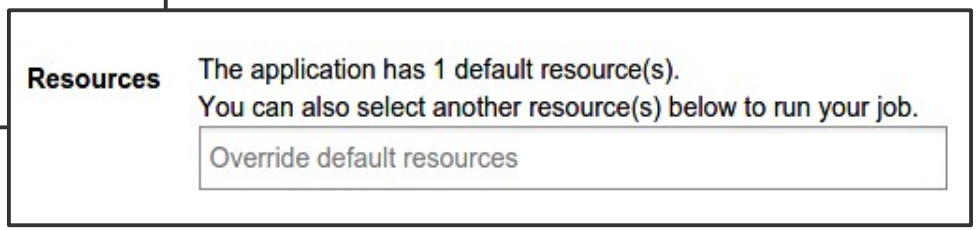
Integration with EGI



Binding Applications to Resources







Resource Binding: Challenges

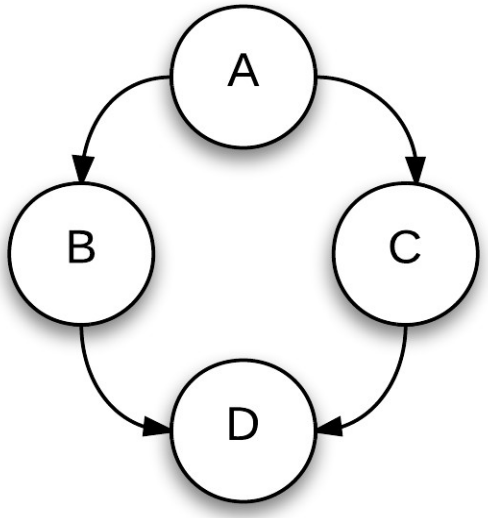
- Dynamic binding
  - Protecting users/resources from malicious/broken code
    - Common practices (trust, code signing, verification, publication)
    - Using virtualization and sandboxing solutions (Docker, Firejail)
  - Making applications portable across resources
    - Run an application in a preconfigured Docker container
    - Build a portable application package (CDE, CARE)
- Binding with multiple resources
  - Sheduling of application tasks across heterogeneous distributed computing resources

Programming Access to Applications

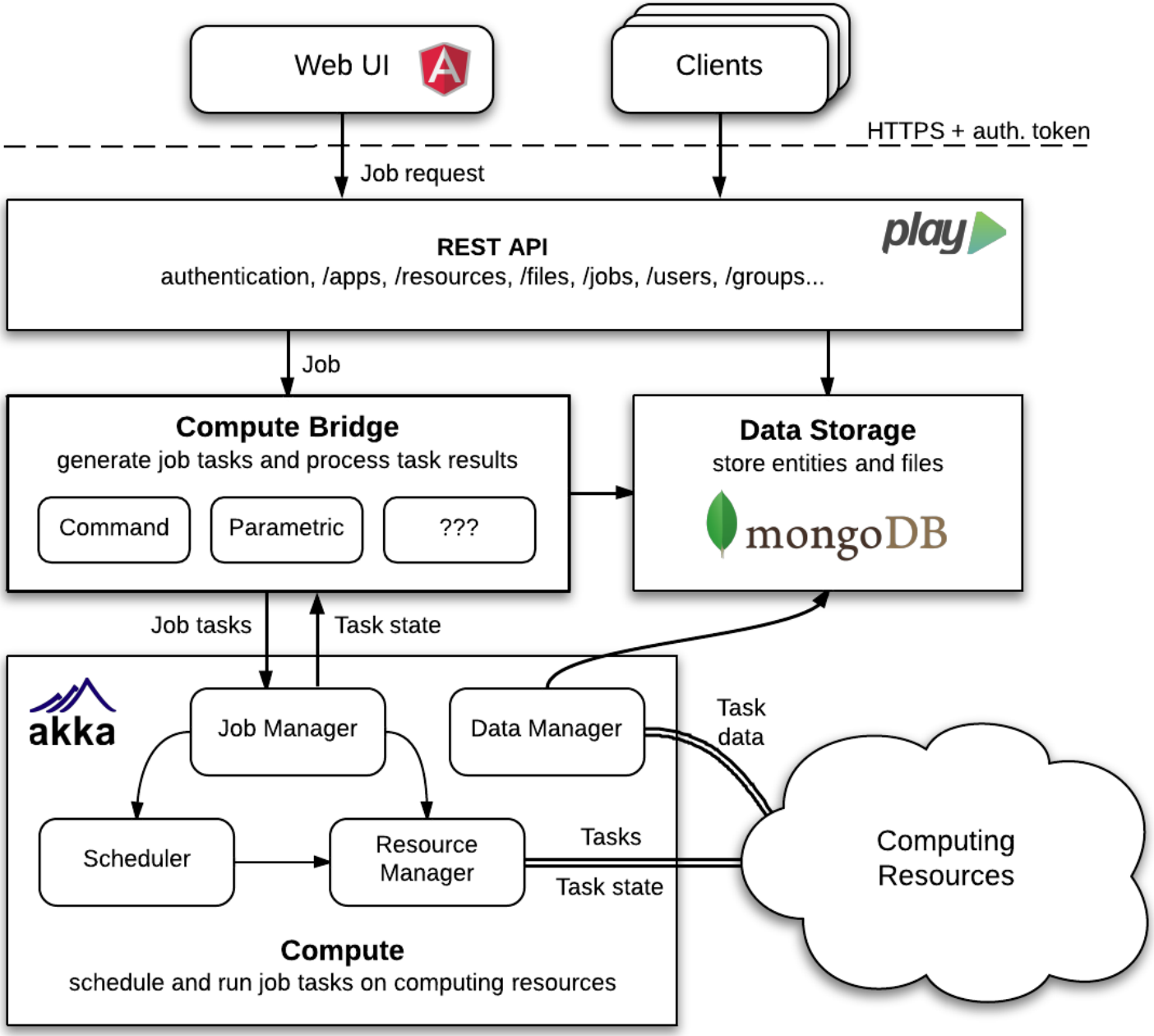
- Why?
  - Automation
    - Repetitve application runs
    - Use of multiple applications (pipelines, workflows)
  - Integration with external systems and third-party tools
- How?
  - Accessing application via web service interface (REST API)
    - HTTP + JSON, any modern programming language
  - Using client library (Python API)
    - Implemented on top of REST API

Python API

```
import everest

session = everest.Session(
  'https://everest.distcomp.org', token = '...'
)

appA = everest.App('52b1d2d13b...', session)
appB = everest.App('...', session)
appC = everest.App('...', session)
appD = everest.App('...', session)

jobA = appA.run({'a': '...'})
jobB = appB.run({'b': jobA.output('out1')})
jobC = appC.run({'c': jobA.output('out2')})
jobD = appD.run({'d1': jobB.output('out'), 'd2': jobC.output('out')})

print(jobD.result())

session.close()
```



Everest Architecture



Experimental Evaluation

- Setup
  - Single server: 2 quad-core Xeon E5620 (2.4 GHz), 24GB RAM, Ubuntu 12.04
  - Applications: Sleep, Autodock Vina, Parameter Sweep
- Raw job submission tests
  - Capable of serving 1000 concurrent clients with acceptable latencies
  - Input file uploads negatively impact throughput and latency
- End-to-end tests (complete job life cycle)
  - Job processing overhead introduced by Everest+agent is 10s of seconds
  - Could be improved to better accommodate short jobs
- Scalability tests
  - 100 agents with 10 slots running in different locations
  - Maximum observed overhead for 1000 jobs is 23 seconds
- Real application runs
  - Ad-hoc grid: 3 servers + 3 clusters (316 cores)
  - Autodock Vina, Parameter Sweep application from geophysics domain

Future Work

- Supporting more complex many-task applications
- Implementing interaction with a running application
- Integration with other types of computing resources
- Optimization of data transfer
- Efficient scheduling of applications across multiple resources
- Improving performance and scalability

Conclusion

- A web-based platform supporting scientific computing
  - Enables users with minimal skills to publish and share scientific applications as services
  - Executes applications on external resources attached by users
  - Implements decoupling of published applications from resources
  - Supports programmatic access to the platform's functionality

- More information: http://everest.distcomp.org/