TWO SIGMA

www.twosigma.com

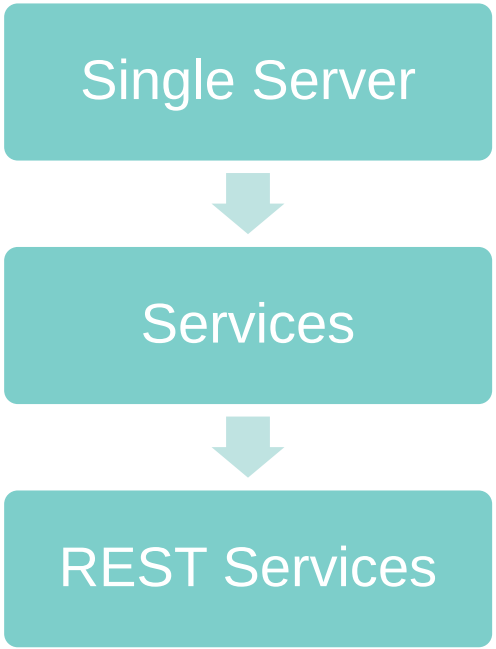
Principles of REST API Design

Presented By: Amy Wasik

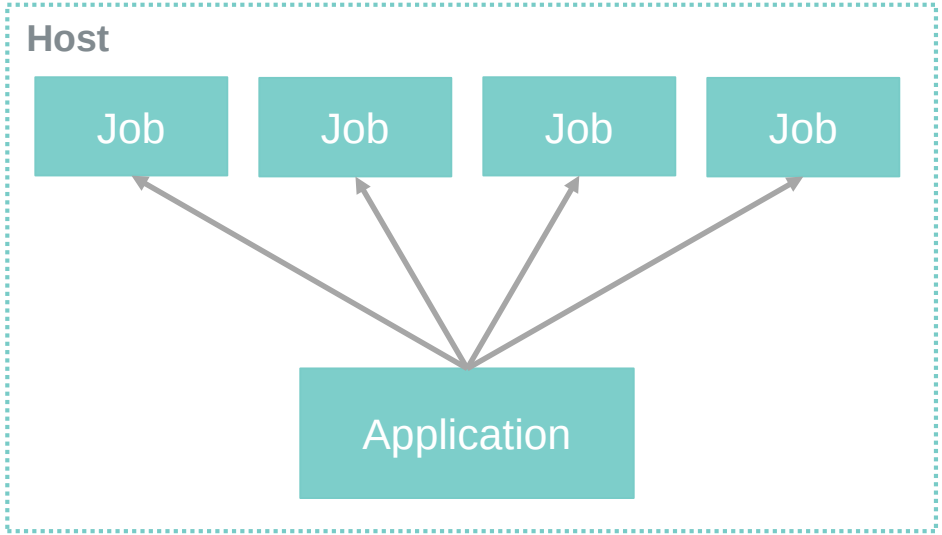
Two Sigma Investments, LP

May 19, 2017

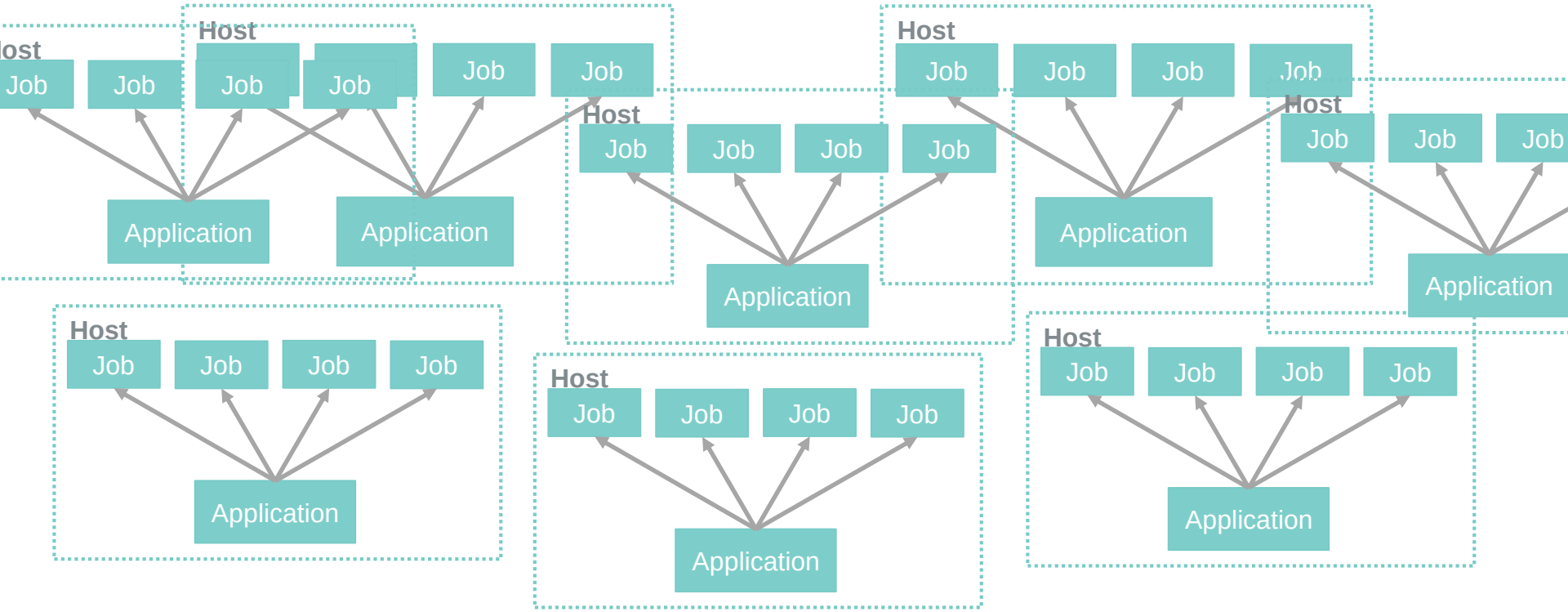
Software Architecture



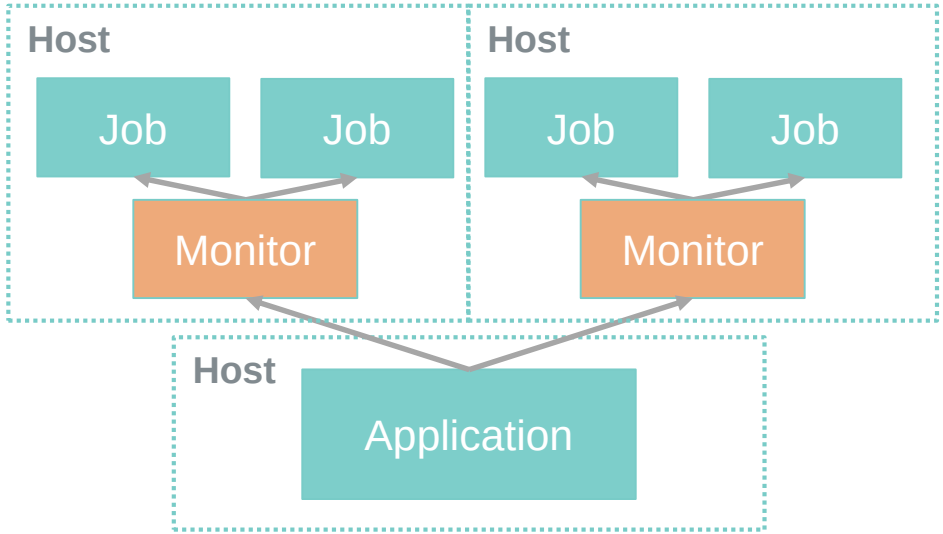
Our Application



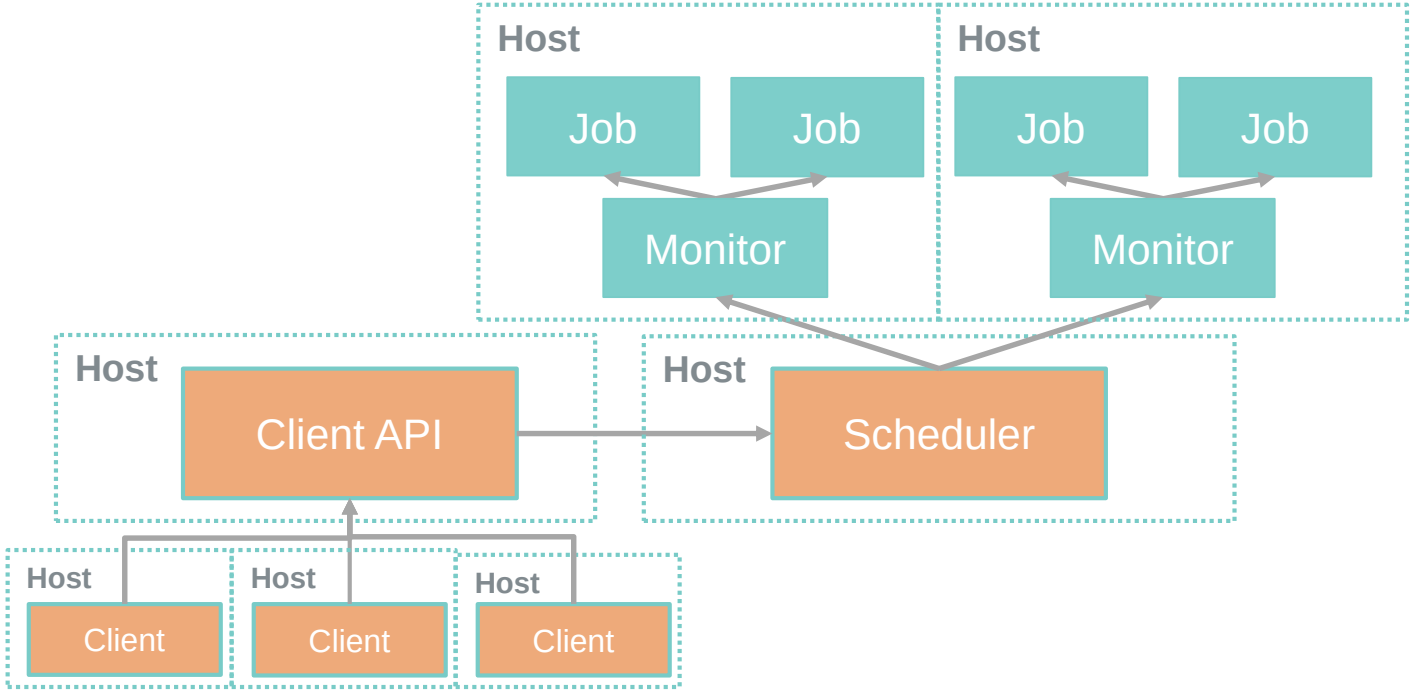
Our Application



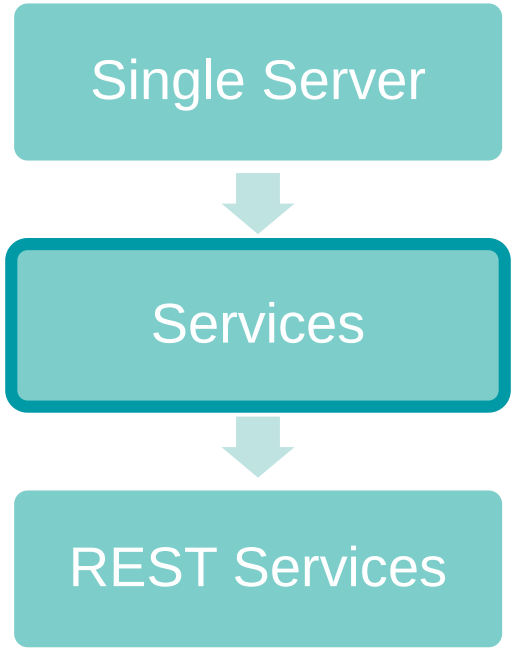
Our Application



Our Application



Software Architecture

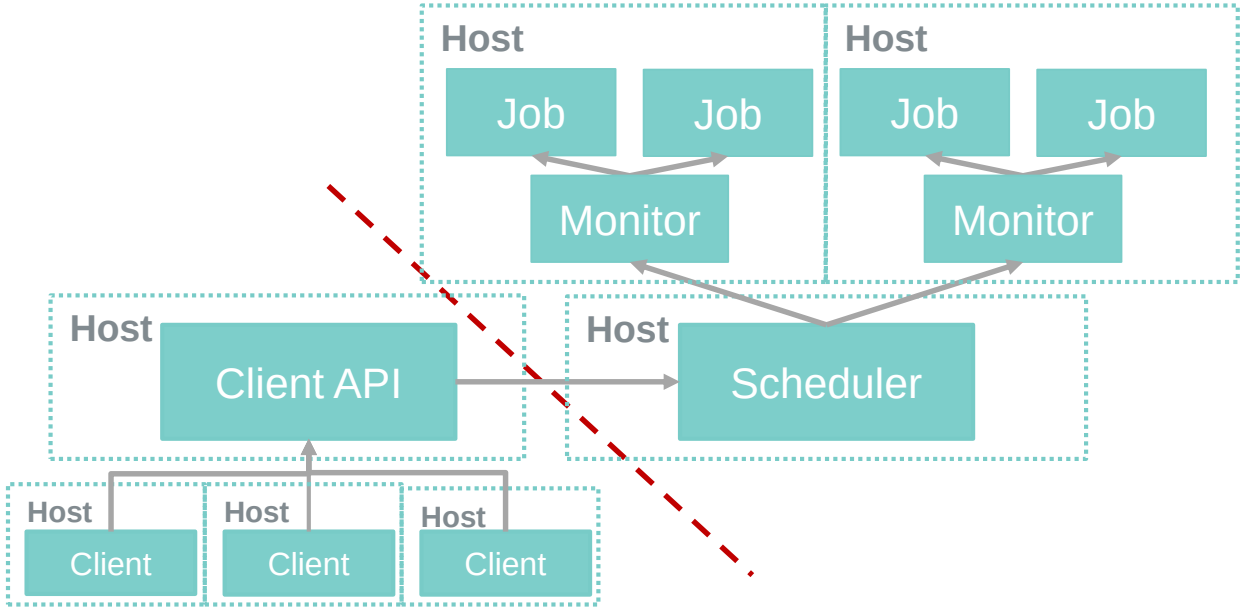


Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient

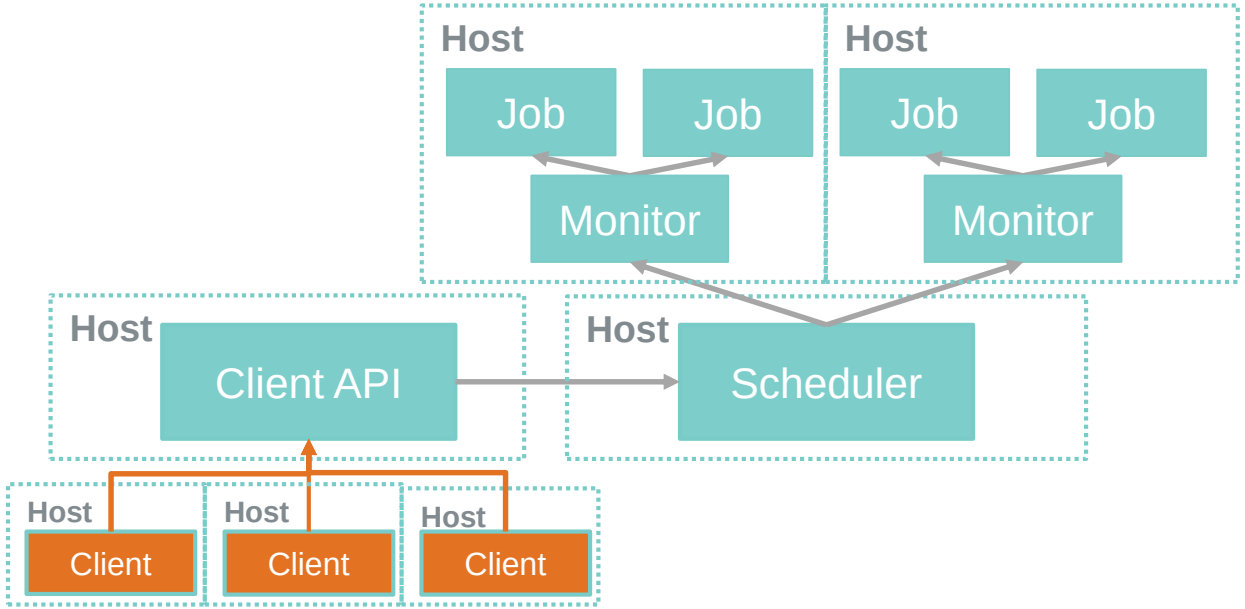
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient



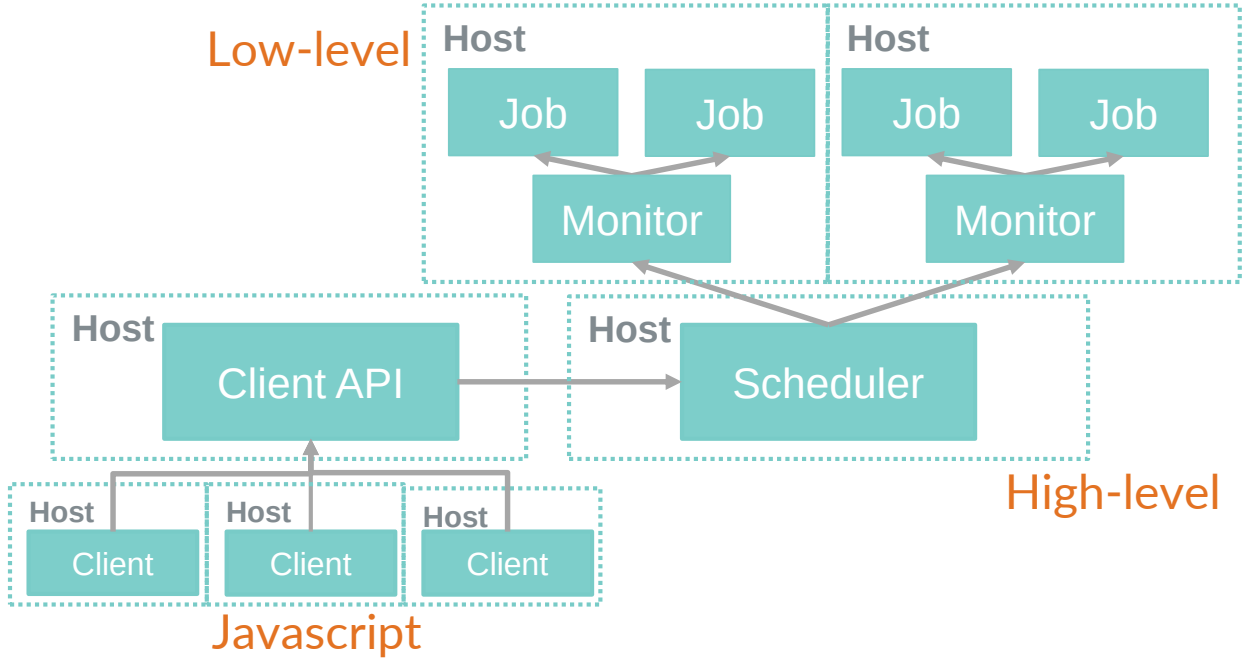
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient



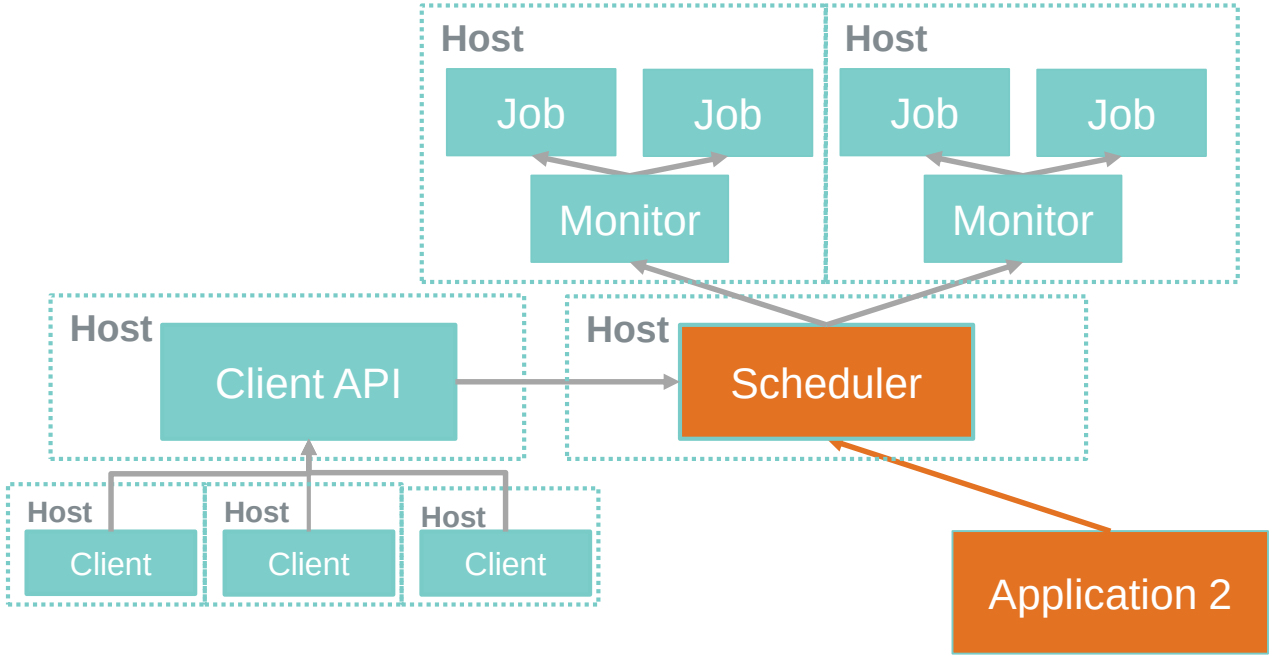
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient



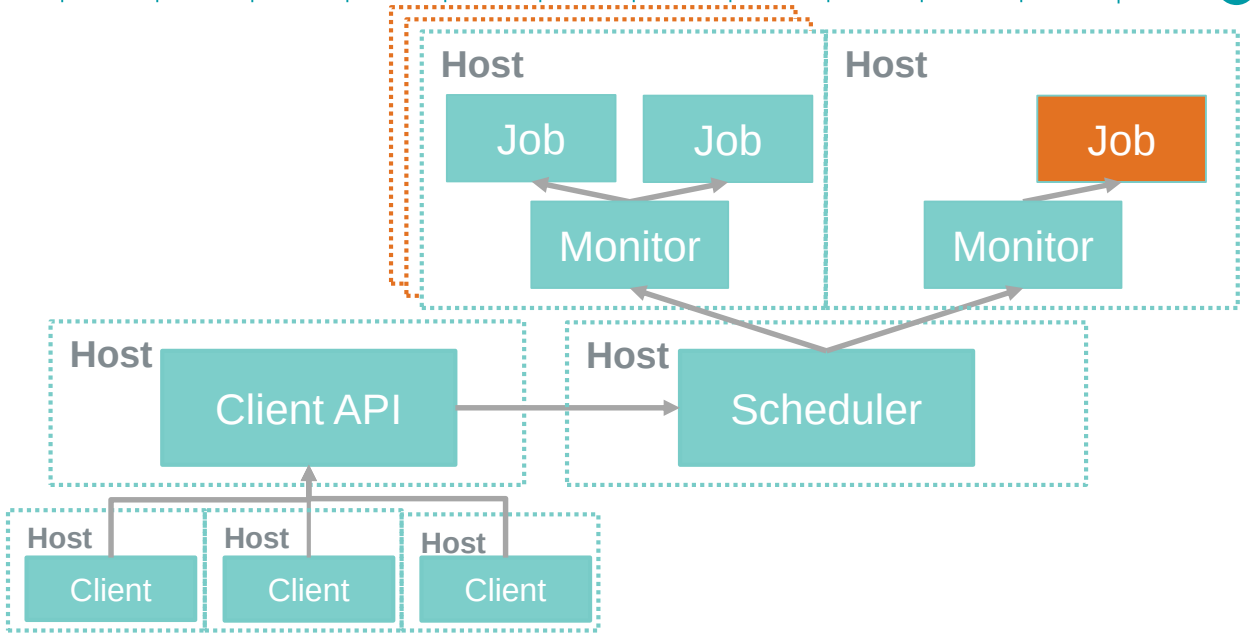
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient



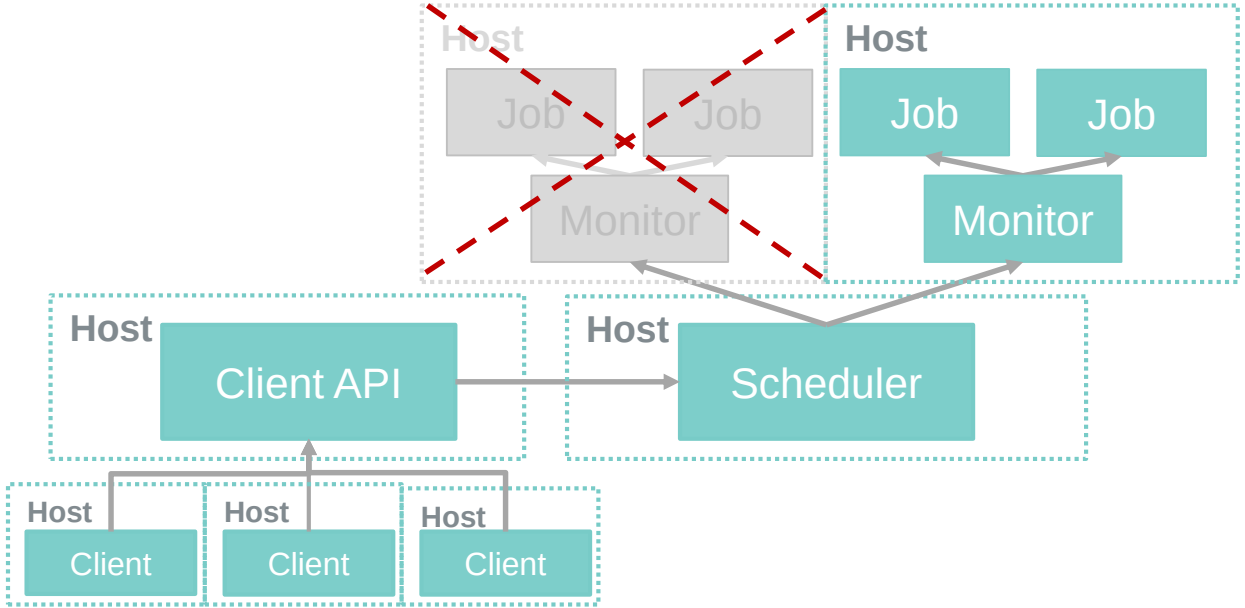
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient



Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient



Communication

Single Server:

Local API Method Calls

Services:

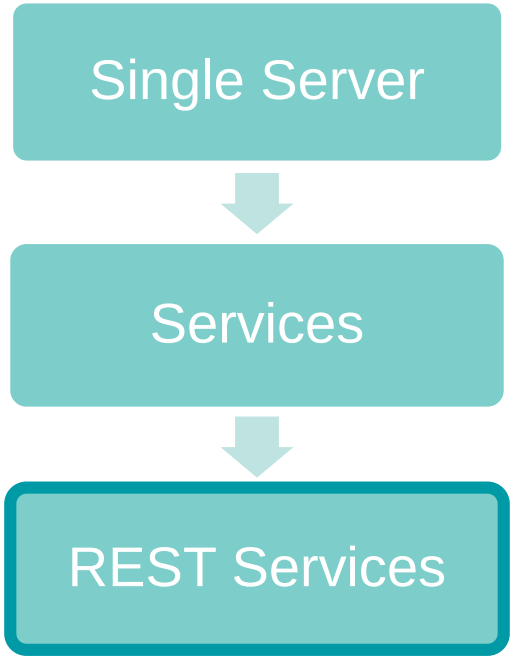
Inter-process Communication (IPC)

Inter-Process Communication (IPC)

- Remote Procedure Calls (RPC)
  - Use a library to convert local calls to remote ones

```
public interface Jobsystem {
  Job createAJob(JobDetails details);
  void submitJob(Job j);
  List<Job> getJobs(String namePattern);
  List<Job> getMyJobs(String user);
  List<Job> getJobsOther(String query);
  Job getJob(int id);
  void updateJob(JobDetails details);
}
```

Software Architecture



REST Architecture

- Representational State Transfer (REST)

Additional Constraints	Benefits
Stateless	Scalability

REST Architecture

- Representational State Transfer (REST)

Additional Constraints	Benefits
Stateless	Scalability
Cacheable	Increased Capacity

REST Architecture

- Representational State Transfer (REST)

Additional Constraints	Benefits
Stateless	Scalability
Cacheable	Increased Capacity
Layered	Low Coupling/Interoperability

REST Architecture

- Representational State Transfer (REST)
  - API Constraints

4 Levels of Adherence	Benefits
0 – HTTP Transport	
1 – Resource Oriented Design	
2 – HTTP Verbs as actions on resources	
3 – Hypertext as the Engine of Application State (HATEOAS)	

https://martinfowler.com/articles/richardsonMaturityModel.html

REST Architecture

- Representational State Transfer (REST)
  - API Constraints

4 Levels of Adherence	Benefits
0 – HTTP Transport	Standard Interface
1 – Resource Oriented Design	Easier-to-Use API
2 – HTTP Verbs as actions on resources	Complete API
3 – Hypertext as the Engine of Application State (HATEOAS)	Easy-to-Learn API

https://martinfowler.com/articles/richardsonMaturityModel.html

REST Level 0

HTTP Transport

- Readable object encoding (typically JSON)
- Standard URI format

Level 0: RPC over HTTP

```
public interface Jobsystem {
  Job createAJob(JobDetails details);
  void submitJob(Job j);
  List<Job> getJobs(String namePattern);
  List<Job> getMyJobs(String user);
  List<Job> getJobsOther(String query);
  Job getJob(int id);
  void updateJob(JobDetails details);
}
```

```
GET http://example.com/createAJob?name=t&...
GET http://example.com/submitJob?id=123
GET http://example.com/getJobs?name=test
GET http://example.com/getMyJobs?user=me
GET http://example.com/getJobs?query=...
GET http://example.com/getJob?id=123
GET http://example.com/updateJob?id=123&
```

Level 0: RPC over HTTP

```
GET http://example.com/createAJob?name=t&user=userA...
HTTP/1.1 200 OK
[other headers]
{ "id": 123
}

GET http://example.com/submitJob?id=123
HTTP/1.1 200 OK
[other headers]
{ "error" : "no permission"
}
```

REST Level 1

Resource Oriented Design

- Divide and conquer
- Easy to understand and navigate API

Standard URI Format

- /{resource}
- /{resource}/{resource-id}
- /{resource}/{resource-id}/{sub-resource}
- /{resource}/{resource-id}/{sub-resource}/{sub-resource-id}

Object Oriented Design

```
GET http://example.com/createAJob?name=
GET http://example.com/submitJob?id=12
GET http://example.com/getJobs?name=
GET http://example.com/getMyJobs?user=
GET http://example.com/getJobs?query=
GET http://example.com/getJob?id=123
GET http://example.com/updateJob?id=123
```

```
GET http://example.com/jobs/create?name=t&user=me

GET http://example.com/jobs/get?name=test
GET http://example.com/jobs/getMy?user=me
GET http://example.com/jobs/get?query=...
GET http://example.com/jobs/123
GET http://example.com/jobs/123/update?name=t2...
GET http://example.com/jobs/123/instances/start
GET http://example.com/jobs/123/instances
```

REST Level 2

HTTP Verbs Represent Actions

- More complete and structured APIs

Common Verbs

- GET – Read (Nullipotent)
- PUT – Update (Idempotent)
- POST - Create
- DELETE – Remove (Idempotent)

REST Level 2

Standard HTTP Response Codes

- Standard results of actions

Success		Client Error		Server Error	
200	OK	400	Bad Request	500	Internal Server Error
201	Created	401	Unauthorized (authentication failure)		
204	No Content	403	Forbidden (not allowed access)		
		404	Not Found		

HTTP Verbs for Actions

```
GET http://example.com/jobs/create?name=t&user=me


GET http://example.com/jobs/get?name=test
GET http://example.com/jobs/getMy?user=me
GET http://example.com/jobs/get?query=...
GET http://example.com/jobs/123
GET http://example.com/jobs/123/update?name=t2...

GET http://example.com/jobs/123/instances/start
GET http://example.com/jobs/123/instances
```

```
POST http://example.com/jobs
  -d '{"name":"test", "user": "me", …}'

GET http://example.com/jobs?name=test
GET http://example.com/jobs?user=me
GET http://example.com/jobs?query=...
GET http://example.com/jobs/123
PUT http://example.com/jobs/123
  -d '{"name":"job" …}'
POST http://example.com/jobs/123/instances
GET http://example.com/jobs/123/instances
```

HTTP Verbs for Actions

```
POST http://example.com/jobs
  -d '{"name":"test", "user": "me", …}'
HTTP/1.1 201 Created
[other headers]
{ "id": 123
}

POST http://example.com/jobs/123/instances
HTTP/1.1 403 Forbidden
[other headers]
{ "errorCode" : 10,
  "moreInfo" : "no permission to run this job"
}
```

REST Level 3

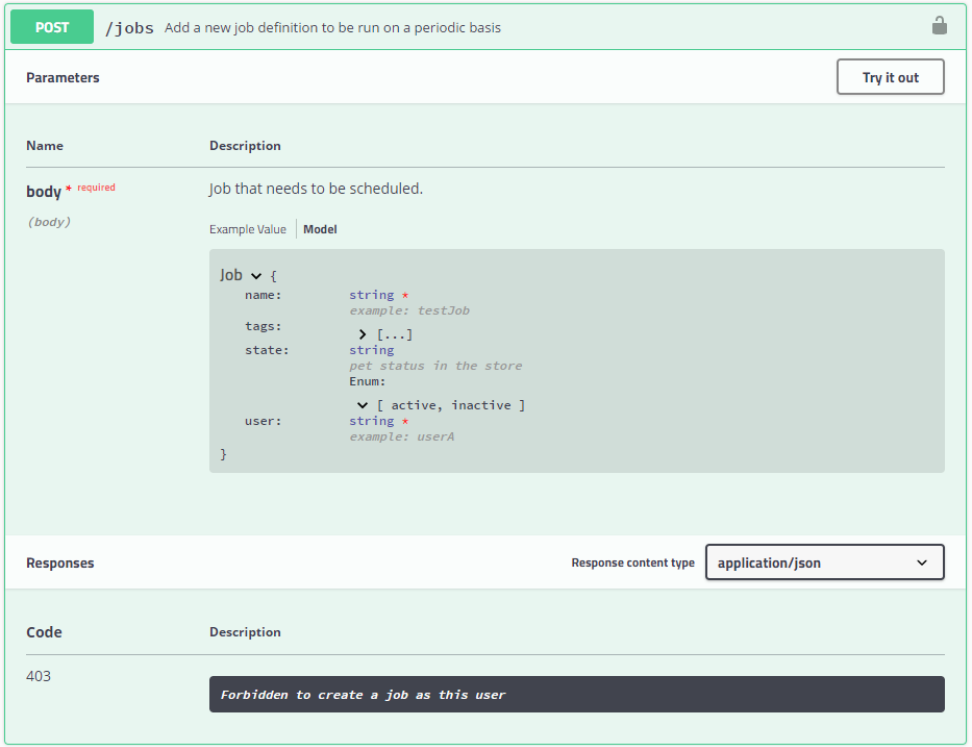
REST API Documentation and API Discoverability

- Hypertext As The Engine Of Application State (HATEOAS)
  - Adds links to response that indicate useful actions
- Open API
  - Provides language-agnostic way to describe REST API
  - Lots of tooling for automation

Open API


jobs Job definitions
POST /jobs Add a new job definition to be run on a periodic basis
GET /jobs Get jobs by query
GET /jobs/{jobId} Find job by jobId
PUT /jobs/{jobId} Update an existing job definition
DELETE /jobs/{jobId} Deletes a job definition
POST /jobs/{jobId}/instances Start running an instance of the job specified by jobId
GET /jobs/{jobId}/instances Get a list of recent instances for a job defined by jobId
PUT /jobs/{jobId}/instances/{instanceId} Change the priority of a running job instance process specified by instanceId
DELETE /jobs/{jobId}/instances/{instanceId} Stop a running insatnace of a job specified by instanceId

Open API


POST
/jobs
Add a new job definition to be run on a periodic basis
Parameters
Try it out
Name
Description
body * required
(body)
Job that needs to be scheduled.
Example Value
Model
Job { }
name: string * example: testJob
tags: [...]
state: string pet status in the store Enum: [active, inactive]
user: string * example: userA
Responses
Response content type
application/json
Code
Description
403
Forbidden to create a job as this user

Evolution of the API

```
public interface Jobsystem {
  Job createAJob(JobDetails details);
  void submitJob(Job j);
  List<Job> getJobs(String namePattern);
  List<Job> getMyJobs(String user);
  List<Job> getJobsOther(String query);
  Job getJob(int id);
  void updateJob(JobDetails details);
}
```

Evolution of the API

Conclusion

- Modern day best practices
  - Services architectures
  - REST APIs
  - Resource Oriented Design
  - Self-documenting code
- Next steps
  - Evolving APIs
  - Complex operations
  - Error handling, Standard response types

Additional Resources

Thank you!

http://swagger.io/

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

https://martinfowler.com/articles/richardsonMaturityModel.html

Published API Guides:

https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

https://github.com/paypal/api-standards/blob/master/api-style-guide.md

https://cloud.google.com/apis/design/