Vendor Neutral Serverless Apps in Python/Zappa

Ram Vedam (@unxn3rd)
Atif Mahmood (@AtifDev)

Agenda

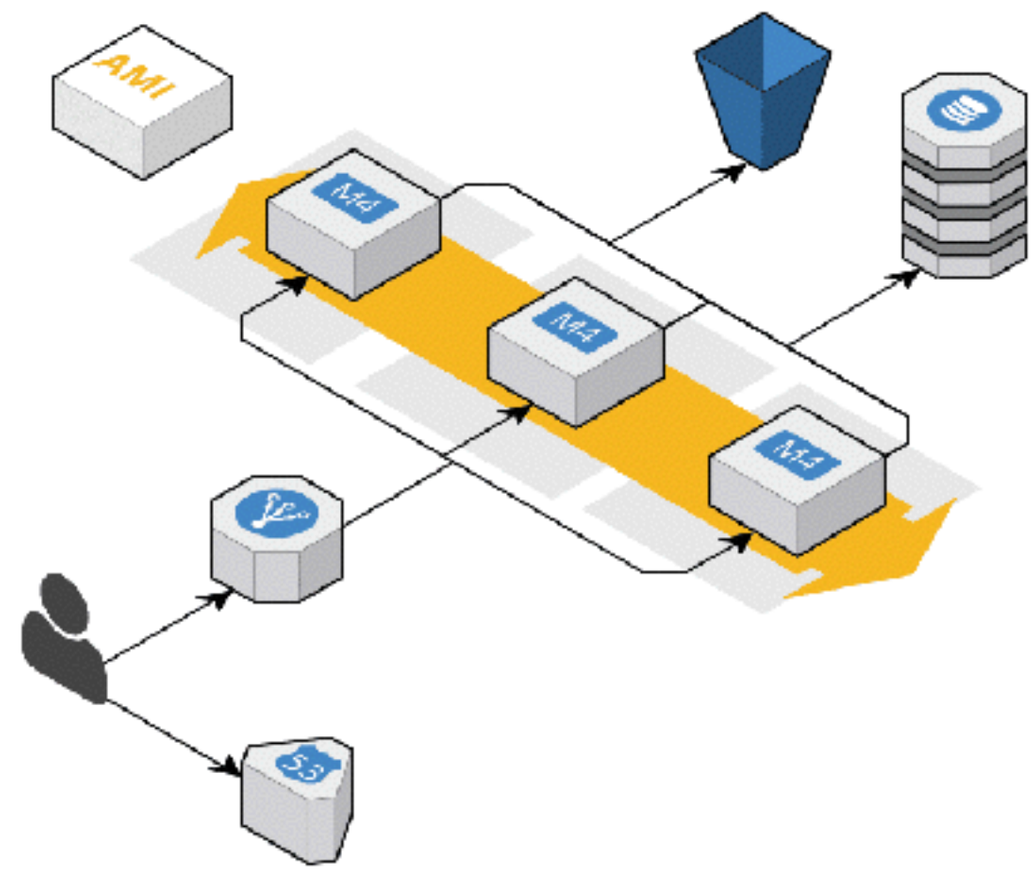
- Definitions
- Serverless vs Traditional
- Why Python + Zappa
- Pitfalls to avoid
- Zappa, Google AppEngine, and WSGI
- Demo deployments
- Advantages and Limitations

Definitions

- SaaS: Your just a user with little responsibility
- PaaS: More on the vendor, less on you
- IaaS: You handle all but the physical hardware
- Monolith: Application with tightly coupled services
- Microservice: Do only one thing, and one thing well.

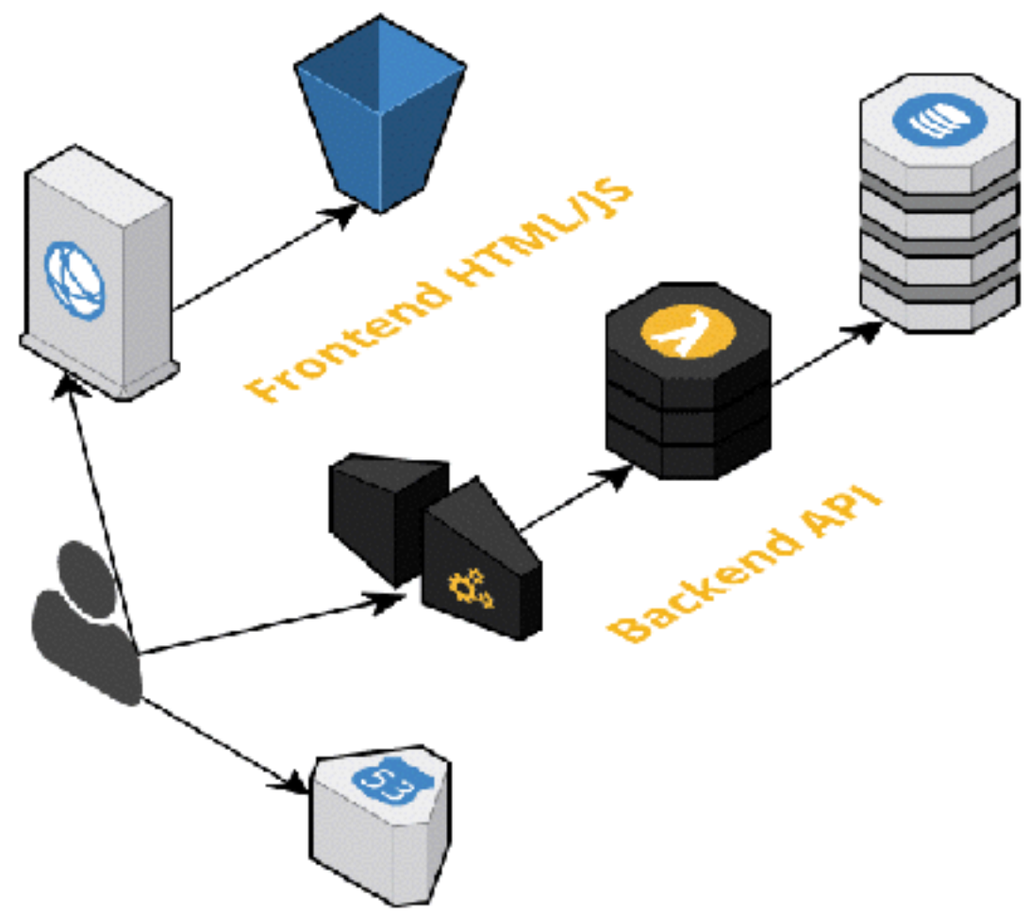
Traditional HA (EB)

- Role based access
- VPC
- Load Balancer
- Autoscale group
- Security groups 2+
- AZ and Regions
- AMI to patch
- Instances to babysit
- Complex Packaging



Serverless

- Serverless Frontend
- Serverless Backend
- Role based access to AWS resources
- Smaller attack surface
- True PaaS



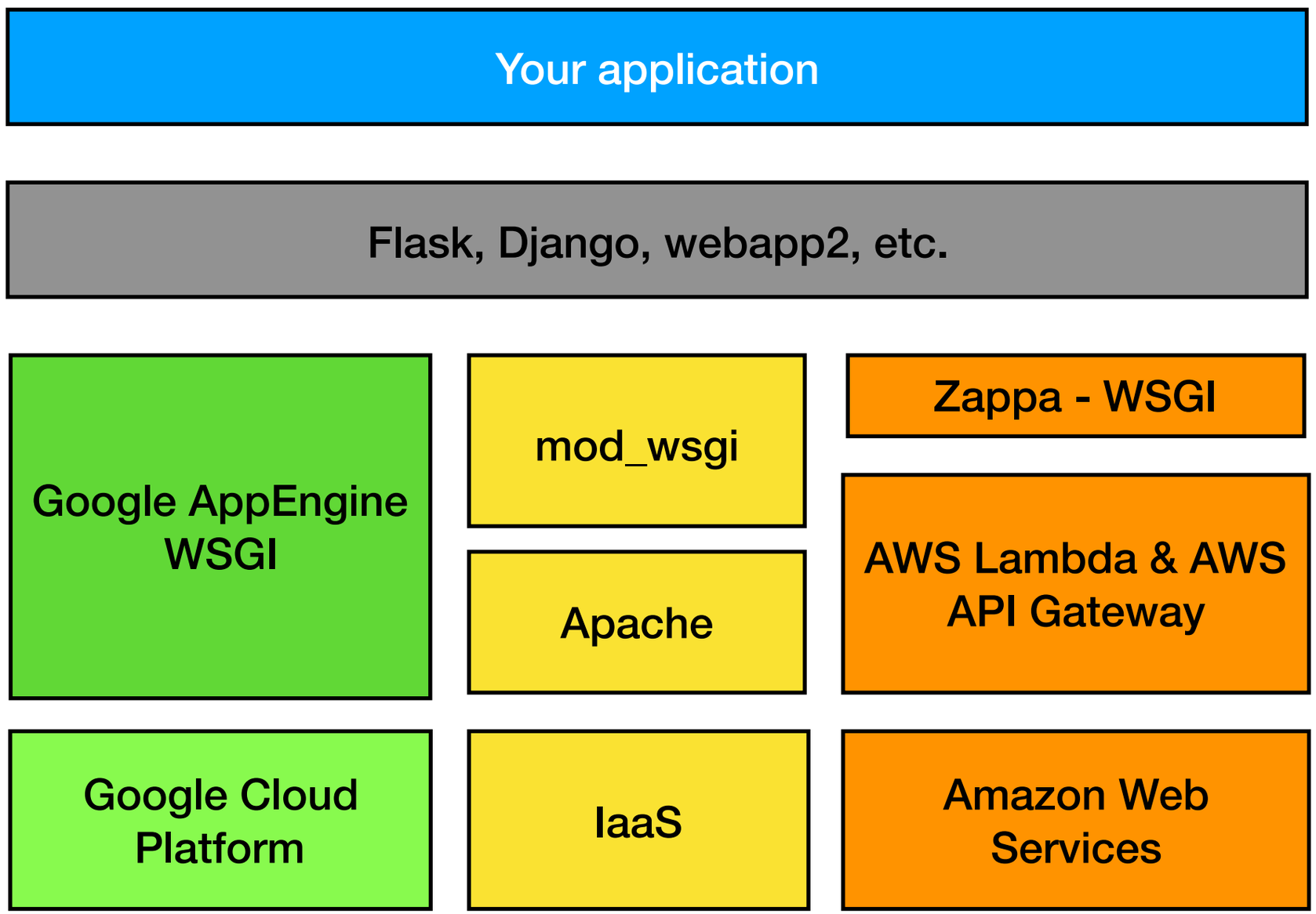
Why Zappa

- Monolith — MicroService — NanoService
- Cold start
- Routing
- Complex deployment
- Console use
- Unit testing
- Difficult Debugging

Suggested Approach

- Build onto of a Python framework (Flask/Django etc)
- Design what is included in the micro-service
- Add zappa_settings.json to deploy on AWS
- Add app.yaml to deploy on Google App Engine
- User a Container or Instance for IaaS deployment

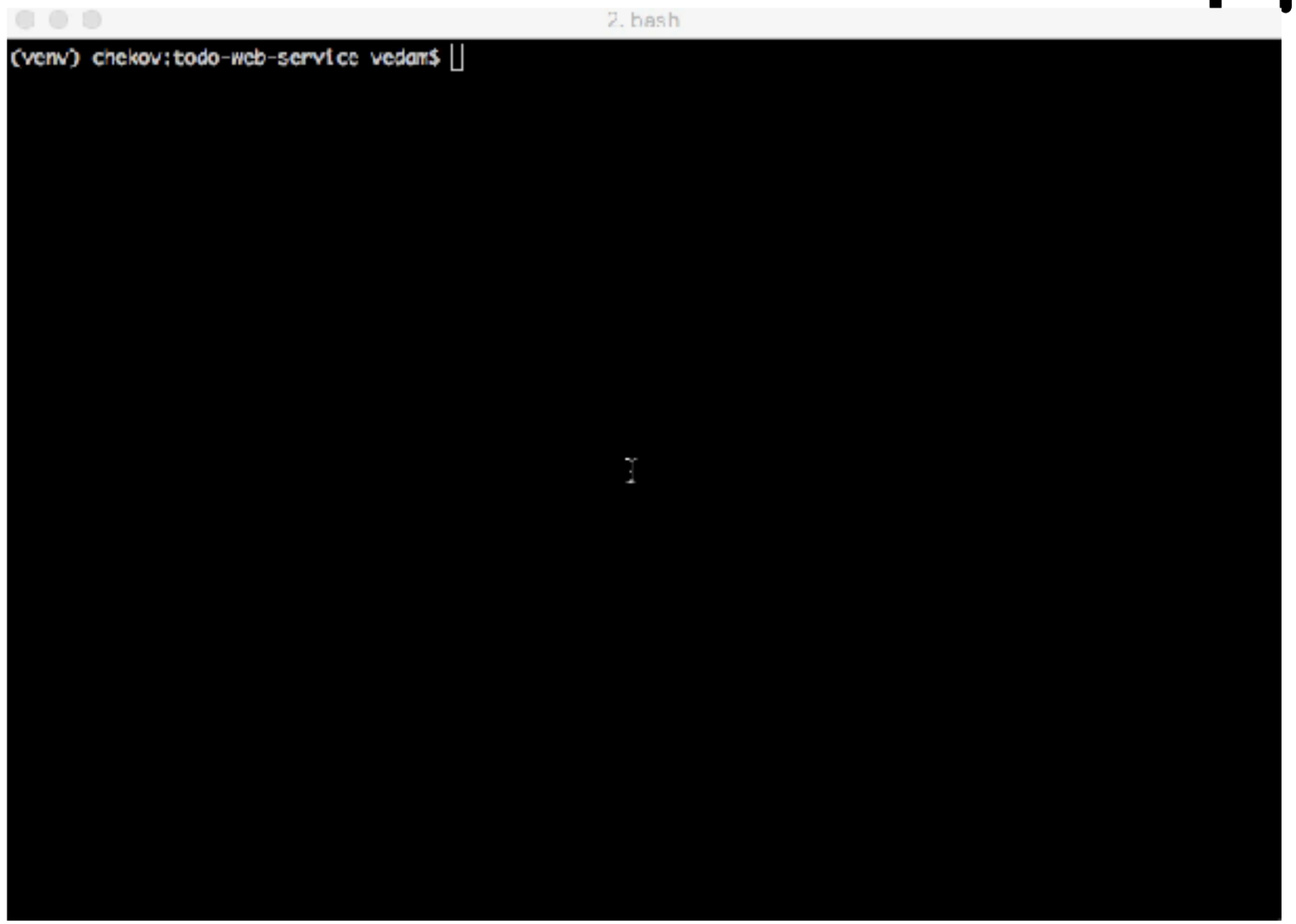
Vendor Neutral Architecture using Python/Zappa



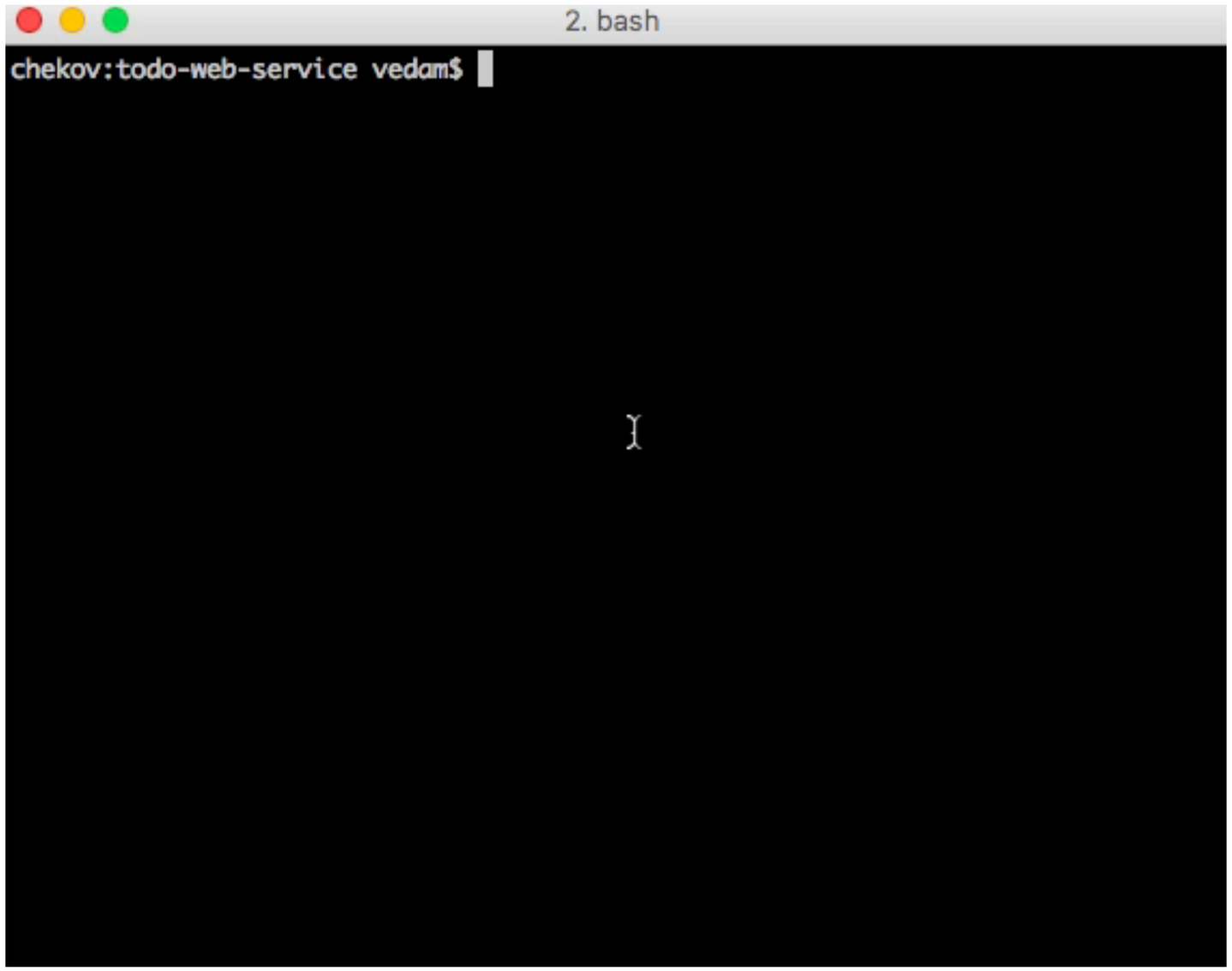
WSGI

- Web Server Gateway Interface
  - Standardized in PEP 3333 in 2010
  - Used as specification for a universal interface for all web applications or frameworks
- Used by many major web frameworks commonly used in Python
  - Django, flask, webapp2, etc.
- https://wsgi.readthedocs.io for more information

Demo
(AWS Lambda with Zappa)



Demo (Google Cloud Platform)



app.yaml

```yaml
runtime: python27
api_version: 1
threadsafe: true
service: default
instance_class: B1
basic_scaling:
  max_instances: 1
  idle_timeout: 5m
env_variables:
  CLOUDSQL_CONNECTION_NAME: todo-web-service-dev:us-east1:todos
  CLOUDSQL_USER: <username>
  CLOUDSQL_PASSWORD: <password>
libraries:
- name: MySQLdb
  version: "latest"
handlers:
- url: /.*
  script: todos.app
```

Gotchas

- Google Cloud Platform
  - can only use Google App Engine Standard for Serverless apps with Python
  - can only use Python 2.7 when deploying on GAE standard
- AWS
  - Using IAM_AUTH with signed s3 bucket url's doesn't work
  - Frequency <50ms requests in succession, expensive

Advantages and Limitations

- Advantages
  - Requires only either a zappa_settings.json (AWS) or a app.yaml (GCP) for deployment
- Limitations
  - Deployable on any cloud platform that supports Python Flask using WSGI (currently only Google Cloud and AWS)
  - Limited to Pure Python dependencies when using Google Cloud AppEngine Standard. There may be alternative libraries based on what you are using

Zappa Serverless Architecture

- No upfront provision
- No Servers to babysit
- Almost no Security Groups
- VPC's optional
- No Auto scale groups
- No Ami's to maintain
- No instances to patch
- Pay per use in 100ms increments

- Reduced time to market
- Reduced Cost
- Invest in the core competency
- Group your code
- Pay for slow start less often
- Debug locally
- Smaller attack surface
- Lower share of responsibility

Questions??

Url: https://github.com/Miserlou/Zappa
slack: slack.zappa.io

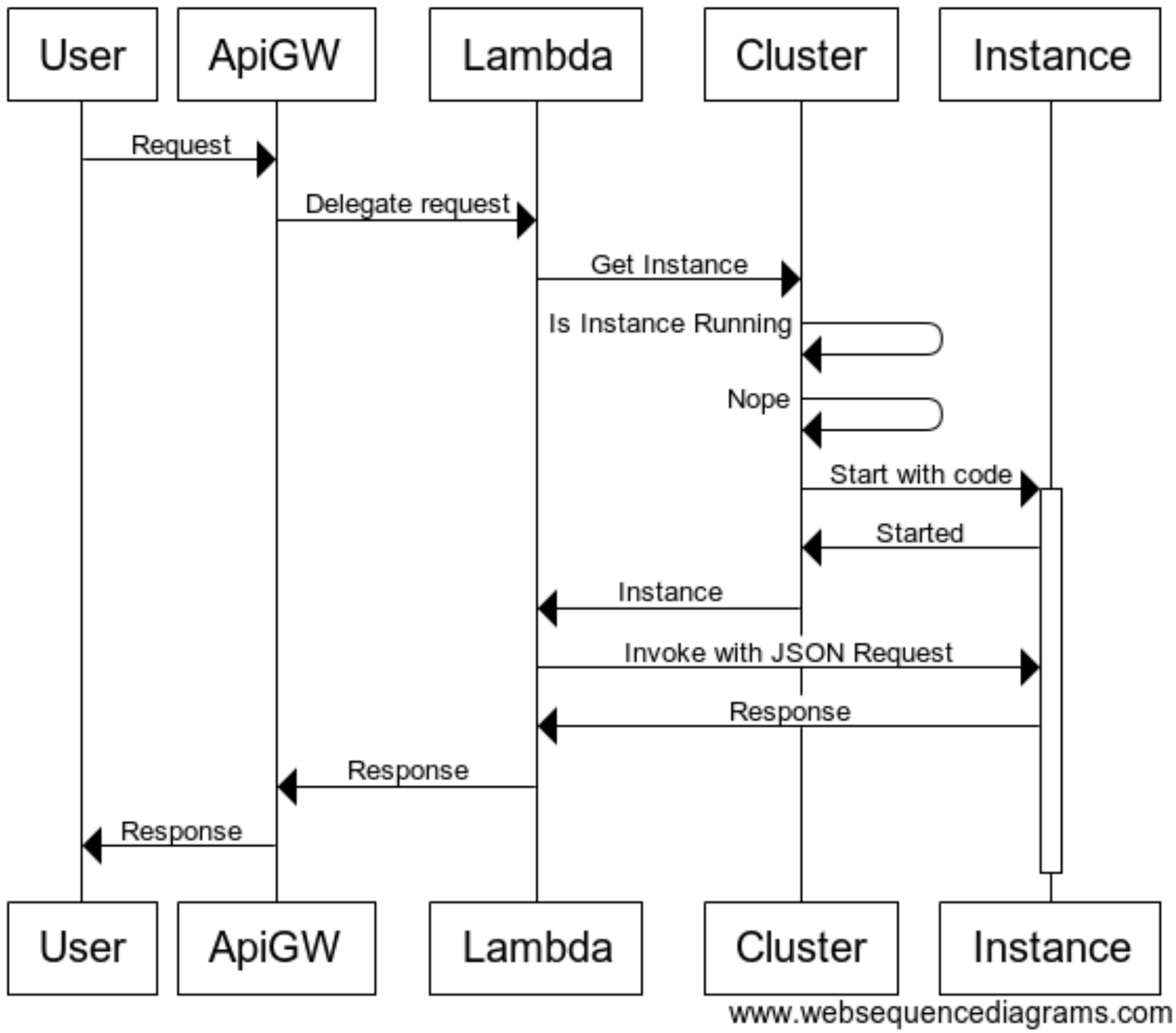
Ram Vedam (@unxn3rd)
Atif Mahmood (@AtifDev)

Coldstart in AWS

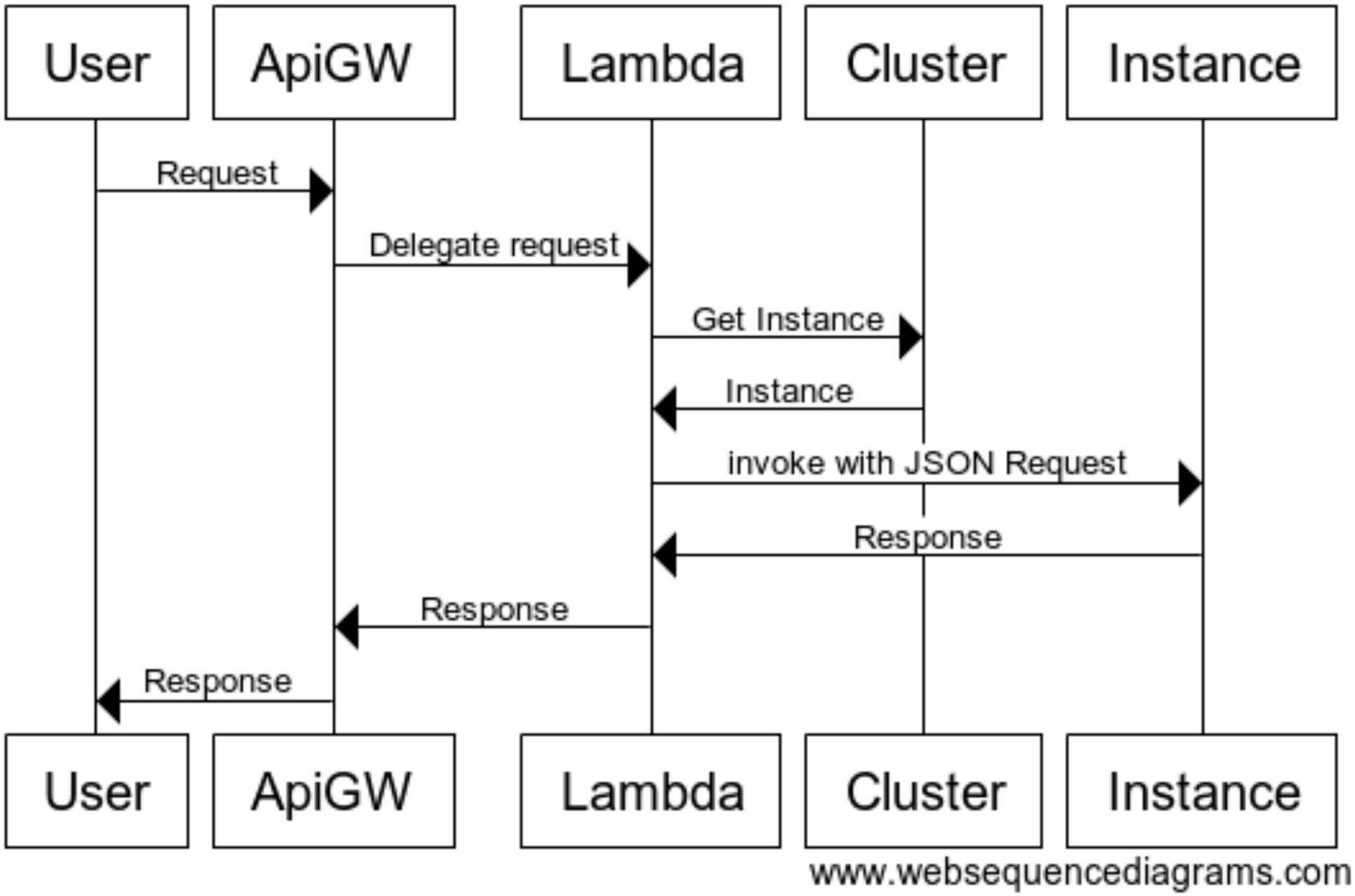
Language	Cold Start Large - Small	Usable Size	Frameworks
Python (2.7/3.6)	0.3 - 3 ms	Any	Zappa (Flask/Django), Chalice
Node 6	2 - 36 ms	Any	ServerLess
Java 8	425ms - 4000ms	1.5G	-
C#	694ms - 5000ms	1.5G	-

Note: Times are before framework initialization. See below for details
https://read.acloud.guru/does-coding-language-memory-or-package-size-affect-cold-starts-of-aws-lambda-a15e26d12c76

Serverless: Cold Start



Serverless


User
ApiGW
Lambda
Cluster
Instance
Request
Delegate request
Get Instance
Instance
invoke with JSON Request
Response
Response
Response
User
ApiGW
Lambda
Cluster
Instance
www.websequencediagrams.com